Customer Driven Project

tdt4290

Group 7:

Bendik Solheim
Brian Mathé
Erik Andreas Larsen
Kjetil Aamodt
Olav Bjørkøy
Riccardo Ortolan

November 18, 2009

Preface

This report is the final product of the course TDT4290 - Customer Driven Project. It is made by 6 students at The Norwegian University of Science and Technology in 2009. The report contains everything produced during the 13 weeks of the project, except the prototype itself.

The task was given by the norwegian company Lingit AS, which develop software solutions that aims to help people with reduced writing and reading abilities. With this in mind, they wanted a prototype of a web based application where people would be able to send in documents and markup texts for either translation, proofreading or both.

We would like to thank our client contact, Tor Rune Skoglund, for his cooperation and support throughout the project. We would also like to thank our supervisors, Basit A. Kahn and Jan M.W. Kristiansen, for valuable input and guidance on this report during the course.

Contents

I Planning & Requirements **2**

1 Project Directive **4**
1.1 Project Purpose 4
1.2 Project Mandate 4

2 Planning **9**
2.1 Project Plan 9
2.2 Organization 13
2.3 Risk Management 15
2.4 Quality Assurance 20

3 Preliminary Studies **23**
3.1 Limitations 23
3.2 Scrum 24
3.3 Version Control 25
3.4 Model - View - Controller 26
3.5 PHP 27
3.6 Code Conventions 32

4 Requirements **34**
4.1 Product backlog 34
4.2 Non-functional Requirements 41
4.3 Use Cases 43
4.4 Test Plan 60

II Sprints **67**

5 Sprint 1 **69**
5.1 Sprint Planning 69
5.2 Deliverables 71
5.3 Sprint Evaluation 75

6 Sprint 2 **77**
6.1 Sprint Planning 77

6.2 Deliverables . . . 81
6.3 Sprint Prototype . . . 93
6.4 Client Feedback . . . 93
6.5 Sprint Evaluation . . . 94

7 Sprint 3 **96**
7.1 Sprint Planning . . . 96
7.2 Deliverables . . . 99
7.3 Sprint Testing . . . 111
7.4 Sprint Prototype . . . 113
7.5 Client Feedback . . . 113
7.6 Sprint Evaluation . . . 114

8 Sprint 4 **116**
8.1 Sprint Planning . . . 116
8.2 Deliverables . . . 118
8.3 Sprint Testing . . . 134
8.4 Sprint Prototype . . . 135
8.5 Client Feedback: . . . 135
8.6 Sprint Evaluation . . . 136

9 Sprint 5 **138**
9.1 Sprint Planning . . . 138
9.2 Deliverables . . . 140
9.3 Sprint Testing . . . 143
9.4 Sprint Prototype . . . 146
9.5 Client Feedback . . . 146
9.6 Sprint Evaluation . . . 147

III Conclusion & Evaluation **149**

10 Overall System Architecture **151**
10.1 Overall Architecture . . . 151
10.2 Further Development . . . 155
10.3 Limitations . . . 160

11 Project Evaluation **161**
11.1 Using Scrum . . . 161
11.2 Group Dynamics . . . 161
11.3 Learning PHP and CakePHP . . . 162
11.4 Risks . . . 163
11.5 Time Estimation . . . 163

IV Definitions, References & Appendices **166**

12 Acronyms **167**

13 Glossary **170**

A Project Directive Appendix **i**
A.1 Templates i
A.2 Test Case Descriptions iii

B Sprint 1 Appendix **i**
B.1 Wireframes i

C Sprint 2 Appendix **i**

D Sprint 3 Appendix **i**

E Closure Appendix **i**
E.1 Screen shots i

F Manuals **i**

List of Figures

2.1 Gantt Chart - Project Schedule 11
2.2 Organization Chart 13

3.1 Scrum Project Lifecycle 24
3.2 CakePHP MVC Request [8] 32

4.1 High Level System Diagram 35
4.2 Overall Use Case Diagram 43
4.3 Create Account Use Case 44
4.4 Customer Login Use Case 44
4.5 Previous Documents Use Case 45
4.6 Documents in Progress Use Case 46
4.7 Buy More Words Use Case 46
4.8 Specify Preferences Use Case 47
4.9 Document Proofread Use Case 48
4.10 Input Text Use Case 49
4.11 Upload Job Use Case 49
4.12 Contact Administrator Use Case 50
4.13 Language Selection Use Case 51
4.14 Mail Notification Use Case 51
4.15 Download Results Use Case 52
4.16 SMS Notification Use Case 52
4.17 Employee Login Use Case 53
4.18 Upload Finished Document Use Case 54
4.19 Take Jobs Use Case 55
4.20 Download Documents Use Case 55
4.21 Receive Mail About New Tasks Use Case 56
4.22 Administrator Login Use Case 56
4.23 Add New Employee Use Case 57
4.24 See All User Details Use Case 58
4.25 Delete Users Use Case 58
4.26 Add New Administrator Use Case 59
4.27 Send Mail Use Case 59
4.28 Black-box Testing 61

5.1 High Level System Focus Diagram 70

5.2 Wireframe Homepage . . . 72
5.3 ER Diagram . . . 74
5.4 Burndown Chart Sprint One . . . 76

6.1 High Level System Focus Diagram . . . 78
6.2 Context Level Data Flow Diagram for the Web Application . . . 78
6.3 Level 1 Data Flow Diagram for the Web Application . . . 79
6.4 MVC Structure for the Document Controller . . . 81
6.5 MVC Model Classes . . . 82
6.6 Upload for Proofread Sequence Diagram . . . 85
6.7 Download Finished Document Sequence Diagram . . . 87
6.8 Use case for the Microsoft Word Macro . . . 88
6.9 Select Text Sequence Diagram . . . 89
6.10 Customer Login Sequence Diagram . . . 90
6.11 Send Text Sequence Diagram . . . 92
6.12 Burndown Chart Sprint 2 . . . 95

7.1 High Level System Focus Diagram . . . 97
7.2 Viewing or Editing a Customers Details . . . 100
7.3 Deleting a Customer . . . 101
7.4 Buy More Words . . . 102
7.5 Show Finished and Not Finished Documents . . . 104
7.6 Set Language . . . 105
7.7 Employee Takes a New Document . . . 106
7.8 Uploading a Document for Proofreading . . . 107
7.9 Document Automatically Assigned to an Employee . . . 109
7.10 Words Counting . . . 110
7.11 Burndown Chart Sprint 3 . . . 115

8.1 High Level System Focus Diagram . . . 117
8.2 MVC Models Classes . . . 119
8.3 ER-diagram . . . 120
8.4 Enter Billing Information in the Database . . . 121
8.5 Contact Admin . . . 122
8.6 Edit Notification . . . 123
8.7 Set Skills for Employees . . . 124
8.8 Set Employee Status to Unavailable . . . 125
8.9 Add Administrator . . . 127
8.10 Microsoft Word Add-in Use Cases . . . 128
8.11 Microsoft Word Add-in Class Diagram . . . 128
8.12 CheckNews Use Case Sequence Diagram . . . 130
8.13 Translate Use Case Sequence Diagram . . . 132
8.14 Burndown Chart Sprint 4 . . . 137

9.1 High Level System Focus Diagram . . . 139

9.2 Translate Sequence Diagram for Add-in 142
9.3 Burndown Chart Sprint 5 148

10.1 Overall Function of the system 151
10.2 Final ER Diagram 152
10.3 Home Page 154
10.4 Data Flow Diagram for the Microsoft Office Word Add-in 155
10.5 Office Ribbon Tab and Buttons for the Commands of the Add-in . . . 155
10.6 Check News Use Case Enabled 156
10.7 Proofread Use Case Enabled 156
10.8 Translate Use Case Enabled 157

11.1 Cake Diagram of Hours Spent 164
11.2 Column Diagram of Hours Spent Each Week 165

C.1 Uploading a Document for Translating i

D.1 Uploading a Document for Translating i

E.1 Admin Page ii
E.2 Translate Page iii
E.3 Translator Page iv
E.4 Translator Page v

List of Tables

1.1 The Client 5
1.2 Client Contacts 5
1.3 Student Group 5
1.4 Supervisors 5

2.1 Planning Phase 10
2.2 Sprints 12
2.3 Roles 14
2.4 Risk Assessment Table 17

3.1 PHP Frameworks Comparison 30

4.1 Web Application Backlog - High Priority Items 36
4.2 Web Application Backlog - Medium Priority Items 37
4.3 Web Application Backlog - Low Priority Items 39
4.4 Add-on Backlog 40
4.5 Non-Functional Requirements 42
4.6 Create Account Use Case Description 44
4.7 Customer Login Use Case Description 44
4.8 Previous Documents Use Case Description 45
4.9 Document in Progress Use Case Description 46
4.10 Buy More Words Use Case Description 46
4.11 Specify Preferences Use Case Description 47
4.12 Document Proofread Use Case Description 48
4.13 Input Text Use Case Description 49
4.14 Upload Job Use Case Description 49
4.15 Contact Administrator Use Case Description 50
4.16 Language Selection Use Case Description 51
4.17 Mail Notification Use Case Description 51
4.18 Download Results Use Case Description 52
4.19 SMS Notification Use Case Description 52
4.20 Employee Login Use Case Description 53
4.21 Upload Finished Document Use Case Description 54
4.22 Take Jobs Use Case Description 55
4.23 Download Document Use Case Description 55
4.24 Receive Mail About New Tasks Use Case Description 56

4.25 Administrator Login Use Case Description 56
4.26 Add New Employee Use Case Description 57
4.27 See All User Details Use Case Description 58
4.28 Delete Users Use Case Description 58
4.29 Add New Administrator Use Case Description 59
4.30 Send Mail Use Case Description 59
4.31 Test Identification 63
4.32 Test Schedule 64
4.33 Test Description Template 65
4.34 Test Execution Template 65

5.1 Backlog Sprint 1 70

6.1 Web Application Backlog - Sprint 2 79
6.2 Add-on Backlog - Sprint 2 80
6.3 Models 82
6.4 Controllers 83
6.5 Libraries 83
6.6 Text Selection 88
6.7 Create Account Description 90
6.8 Send Text 91

7.1 Changes to Product Backlog - Added Items 97
7.2 Changes to Non-functional Requirements 97
7.3 Backlog 98
7.4 Addon 99
7.5 User Registration Test Execution. 111
7.6 Account Settings Test Execution. 112
7.7 Document Controller Function Test. 112

8.1 Web Application Backlog - Sprint 4 117
8.2 Add-on Backlog - Sprint 4 118
8.3 Added Models 119
8.4 Word add-in Class Diagram Description 129
8.5 Check News Description 129
8.6 Translate Description 131
8.7 Proofread Description 133
8.8 Email Notification Function Test Execution. 134
8.9 Employee Availability Test Execution. 134

9.1 Backlog 139
9.2 Add-on Backlog - Sprint 5 140
9.3 Administrator Management Test Execution. 143
9.4 Performance Test Execution. 143
9.5 Customer Usability Test Execution. 144

9.6 Framework Controllers Tests. 144
9.7 Framework Models Tests. 144
9.8 Framework Helpers Tests. 145
9.9 Framework Database Tests. 145
9.10 Client Acceptance Test Execution. 145

A.1 Document Controller Function Test iii
A.2 Email Notification Function Test iv
A.3 Framework Controllers Tests iv
A.4 Framework Models Tests iv
A.5 Framework Helpers Tests v
A.6 Framework Database Tests v
A.7 User Registration and Login Test vi
A.8 Customer Account Settings Test vi
A.9 Administrator Management Test vii
A.10 Employee Availability Test vii
A.11 Performance Test viii
A.12 Customer Usability Test viii
A.13 Client Acceptance Test ix

Abstract

This report presents the development of the product "Lingtrans" for our client, Lingit AS. This task was given to a student group during the course "TDT4290 - Customer Driven Project".

The development team consists of 6 students at The Norwegian University of Science and Technology who are taking this course as part of a masters degree in Computer Science. The purpose of this project is to educate students in how to plan, execute and finish a relatively extensive project for a real client.

The client, Lingit AS, is an IT company based in Trondheim which specializes in developing writing tools for various text editors. Their programs help users translate documents between languages. They also provide programs to aid dyslectics and others who might have a hard time writing a document.

This project is based on a product idea from Lingit. They want a website where users can register, log in, upload documents, and have these documents translated or proofread in a multitude of languages in an easy, quick and secure manner. The website should work in connection with a plugin for text editors through which the user can submit text directly from their document. The website should support notifying customers and employees about activity on the site by email and SMS.

To develop this product it was decided to use the Scrum method. This report details both the planning and implementation of the product, divided by what Scrum describes as Sprints. The product will be implemented in PHP, using a MySQL database and an Apache server, as requested by the client.

The final part of this report concludes the project and evaluates the planning, development and group dynamics experienced throughout the process.

Part I

Planning & Requirements

Introduction

Purpose

The purpose of the Planning & Reqirements is to present the project, its purpose, the scope and the ultimate goal of the project. It strives to give a deeper understanding of the problem, the proposed solution and to provide a set of guidelines for how the project should be executed.

Scope

Project Directive presents the goal, preconditions and involved parties of the project. The **Planning** chapter describes the project plan, organization strategy, delegation of roles, and identified risks tied to this project. **Preliminary Studies** discusses the choice of tools and technologies to use in our solution and presents the relevant background studies performed. The last chapter, **Requirements**, is a detailed description of the system requirements. It explains the system functionality, and provides information on how these requirements are to be tested to assure that they are carried out.

CHAPTER 1

Project Directive

This chapter will present the involved parties, the background for the task we have been given and the external conditions under which the project will be executed. The final part of this chapter concerns the duration of the project.

1.1 Project Purpose

The purpose of this project is to develop and deliver a product based on the client's requirements. The purpose of this report is to give guidelines on how to develop this product, and to give an overview of the work performed and the conditions under which the product is developed.

1.2 Project Mandate

This section presents the basic conditions for the project, including the project name, the involved parties, the project background and its goal. We also present the external conditions under which this product will be developed and how much time we have at our disposal.

1.2.1 Project Name

The client did not provide a name for the finished product. In this report, the product is tentatively named "Lingtrans". This is consistent with the naming scheme used by Lingit AS for their other services.

1.2.2 Involved Parties

The Client

Table 1.1: The Client

Name	*Address*	*Zipcode*	*City*
Lingit AS	Prof. Brochs gt 8A	7030	Trondheim

Client Contacts

Table 1.2: Client Contacts

Name	*Email*	*Position*
Torbjørn Nordgaard	torbjorn@lingit.no	CEO
Tor Rune Skoglund	trs@lingit.no	Developer
Kristian Skarbø	kristian@lingit.no	Developer

Student Group

Table 1.3: Student Group

Name	*Email*	*Phone*
Kjetil Aamodt	kjetiaa@stud.ntnu.no	+4745250140
Bendik Solheim	bendik@stud.ntnu.no	+4799226565
Olav Bjørkøy	olavfrih@stud.ntnu.no	+4797966950
Erik Larsen	erikanla@stud.ntnu.no	+4791161518
Riccardo Ortolan	ortolan@stud.ntnu.no	+4745019531
Brian Mathé	mathe@stud.ntnu.no	+4745765383

Supervisors

Table 1.4: Supervisors

Namme	*Email*	*Phone*
Basit A. Kahn	basit@idi.ntnu.no	+4773550435
Jan M.W. Kristiansen	janmaxim@stud.ntnu.no	

1.2.3 The Client

Lingit AS was founded in 2001 by two NTNU professors - Torbjørn Nordgård and Jon Atle Gulla. The company has close ties to employees at the university, many of whom have stakes in the company.

Lingit's first product was the BusTUC system, which, based on queries in natural language, provides customers with easy access to bus schedules in Trondheim. BusTUC is the basis for Bussorakelet, a service delivered by Trondheim's bus company Team-Trafikk.

The company launched their next product, Lingdys, in 2002. This product is a writing aid program for dyslectics and others having problems writing. A new version was released in 2004, called Lingright, which is a version of Lingdys adapted for writing in english.

Lingit released their first writing aid program for people with a different first language than norwegian in 2007, called Lingcheck. This was Lingdys 3.0, a version adapted for writing in German.

Today, Lingit has 12 employees, most of whom have a background from NTNU.

1.2.4 Project Background

Lingit provides auxiliary writing and reading software products through their website. They want a prototype of a web based application which provides an online service for proofreading and translation of documents both for existing and new customers, provisionally called Lingtrans.

The idea is that registered users can login to the web application and upload documents, or paste text directly, both for proofreading and translation in any language that Lingtrans supports. Lingit will employ translators and proofreaders who log in to the same site to provide the service. If possible, users can download add-ins or plugins for popular work processors which will make it possible to upload text and documents from within the word processors to the service.

Lingtrans will provide e-mail and SMS notifications to the customer when the proofreading or translation is complete.

1.2.5 Project Goal

The project goal defines the final project product.

Prototype: The solution of the task will be a fully usable prototype of the web application which will consist of two parts; Front-end and back-end. Front-end will consist of

a customer interface, a Lingit employee interface an administration interface. Back-end will be the underlying database.

Project report: The solution will be documented in this report with all the details of the projects processes. The report will also contain an installation manual and deployment manual in the appendix for chapter 9 - Overall System Architecture.

1.2.6 Similar Products

One of the first tasks in the project will be to identify online services similar to the envisioned final product. If such services exist, they could be an indicator of how such a service should work, and how people will want to use it.

Another initial task should be to identify existing word processor plugins, similar to the one this project will produce. Such existing plugins could provide an early look at how the finished add-on part of the product should work.

Finally, the team will also have to examine third party companies for the SMS service, including prices, API and other relevant aspects, to get an estimate of how this part of the project should be executed.

1.2.7 External Conditions

People and Companies

The client: One group member should be responsible for communication with the client, primarily by e-mail, and by voice communication if necessary. There should be weekly or bi-weekly meetings with the client throughout the project.

The supervisors: There should be weekly meetings and regular contact with the group supervisors.

The group: See section 1.3.2 for details on each group member.

Computers

- Each team member will use their personal laptop for the project.
- The code will be stored in a Git repository on a server owned by one of the group members.
- The finished product will run on the client's server, powered by Linux.

Other Resources

- Group rooms on the 5th floor of P-15.

1.2.8 Duration

The estimated workload is 312 hours per person for the entire semester. The group consists of six students, which gives a total amount of 1872 hours for the project.

- Project start: 25th of August, 2009.
- Project end and final presentation: 19th of November, 2009.

CHAPTER 2

Planning

This chapter presents the project plan, schedule, organization, risk assesments and quality assurance.

2.1 Project Plan

The total workload for the project is 13 weeks. The first two weeks will be used for planning, project management and organization. The third week, sprint one, will be used to develop some basic wireframes for the web application. The next 8 weeks will be divided into four more sprints where we will develop our prototype into a fully working product. The last weeks will be used for project closure and preparation for the presentation. Each sprint lasts for two weeks, with an exception of the first one.

2.1.1 Planning

Electing the system requirements and becoming organized started after the first client meeting at Tuesday 25.09.09. Table 2.1 list the tasks we performed the first two following weeks of the project.

Table 2.1: Planning Phase

Description
Setting up an apache server
Getting familiar with auxiliary software tools we will use; Git and LaTex
Brainstorming
Planning of the project
Delegation of roles
Getting to know each other
Product backlog
Use case

We will have one meeting with the client in the end of this period to talk with the client about how we see the product and discuss certain points that are found during the brainstorming.

2.1.2 Development model

The group has decided to use the Scrum method (See section 3.2). In the first client meeting, the client did not have an exact plan for the project. They did not know exactly what they wanted and asked us to brainstorm about possible solutions for their problem. This is why we agreed that Scrum is a good model for us to follow. This is also why we do not have more detailed time estimation of the project. It will be very hard to estimate over all the features we still do not know of. After each sprint we will end up with a new prototype of the system, which the client can then test and comment. This way they can pay close attention to the development and interrupt if needed. Figure 2.1 shows the high-level project schedule.

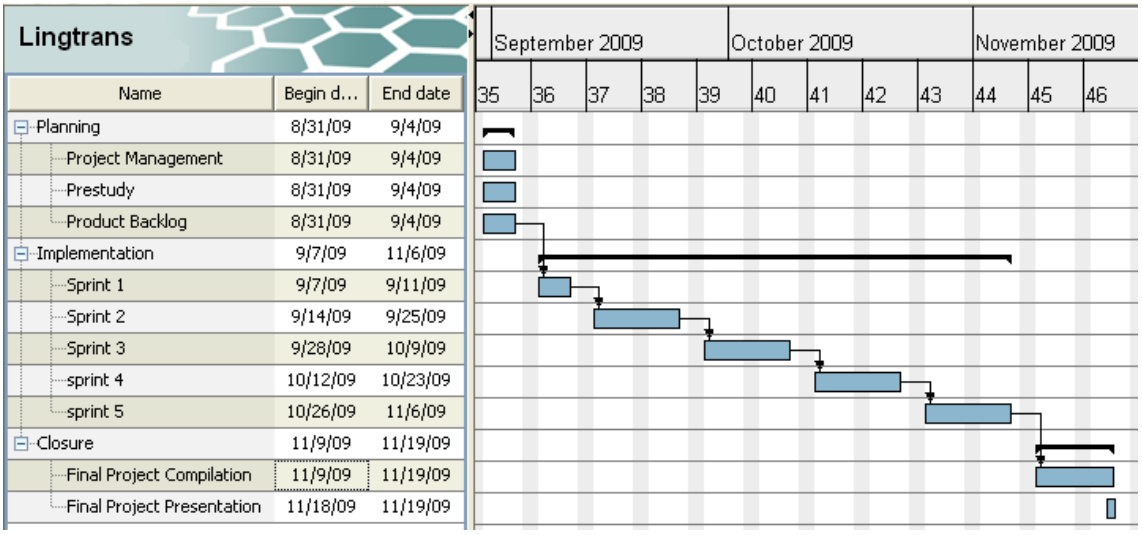


Figure 2.1: Gantt Chart - Project Schedule

2.1.3 Sprints

The total workload, as given from the project description, is 1872 hours, and our plan is to divide this into five sprints and a start phase and end phase. Each sprint will last two weeks, except the first one which will last one week. This suits the client since they wanted to have a meeting every two weeks. The plan for the first sprint is to have wireframes for the most important functionality of the web application, and an ER-diagram of the database. This will give the client a visualization of our solution and will give them the opportunity to provide feedback at an early stage.

For every sprint we will have a sprint planning meeting with the group where we agree upon the deliverables of the sprint. We will have a meeting with the client where we explain to them what our plan for the sprint is. This meeting will in practice be the same meeting as we show what we have done the last sprint because we will only have one meeting with them every two weeks. We will talk with the client after each sprint to discuss the progress and problems and receive feedback on our work. After this meeting we will have another meeting with our group where we discuss how the sprint went and write an evaluation. See table 2.2

Table 2.2: Sprints

Sprints	**Description**	**Time**
Sprint 1	Implementation(wire frames) Database planning/diagram class diagram	Week 37
Sprint 2	Sprint planning meeting Sprint Backlog Implementation Testing Documentation Sprint review meeting	Week 38 and 39
Sprint 3	Sprint planning meeting Sprint Backlog Implementation Testing Documentation Sprint review meeting	Week 40 and 41
Sprint 4	Sprint planning meeting Sprint Backlog Implementation Testing Documentation Sprint review meeting	Week 42 and 43
Sprint 5	Sprint planning meeting Sprint Backlog Implementation Testing Documentation Sprint review meeting	Week 44 and 45
Closing Sprint	Final report compilation Final project presentation	Week 46 and 47

First Sprint

In the first sprint we will make a prototype consisting of wireframes and further develop the items for the product backlog. In the end of the sprint we will show the client our solution of the web interfaces and use the feedback in the next sprint.

Second to Fifth Sprint

The following four sprints will be regular sprints. Between the sprints there may be some changes in the requirements, so in the start of each of these sprints we will have a

sprint planning meeting to establish a sprint backlog and agree on which items of the product backlog to implement. After the review of the requirements, we will start to implement the new functionality, and eventually test it. At the end of the sprints we will have a sprint review meeting, where we will meet the client and show the results of the sprint. We will discuss improvements and suggestions for what to do in the next sprint. Internally in the group we will reflect on how the sprint has been, what went well and what can be improved for the next sprint.

2.2 Organization

This section presents the roles and the main responsibilities for each of the team members during this project. Figure 2.2 shows the relations between the involved parties for this project.

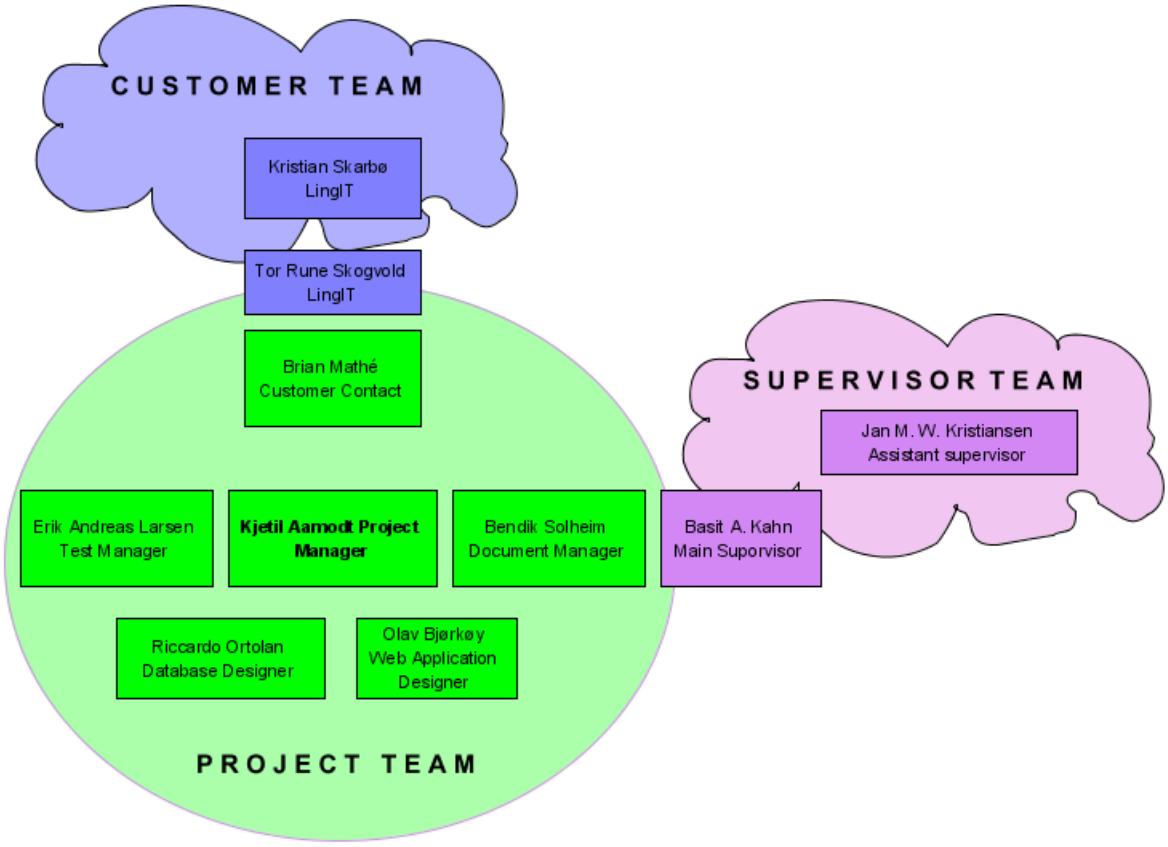


Figure 2.2: Organization Chart

2.2.1 Roles

Table 2.3: Role(Person)

Role	*Tasks*
Project Manager: Kjetil Aamodt	• Main responsibility for the quality of the project • Prepare agendas for meetings • Make sure everyone has something to do and are doing it
Document Manager: Bendik Solheim	• Responsible for creating templates and standard for documents • Main responsibility for the final report content and compilation • Makes sure that all written documents are correct according to templates and standards • Write minutes from meeting for the supervisor meetings • Send meeting minutes to main and assisting supervisors • Ensure that all documents are available to all team members
Client Contact: Brian Mathé	• Arrange appointments with the client • Write meeting minutes • Send meeting minutes and status rapport to the client including a copy to the main supervisor and assisting supervisors • Forward any mail received from the client to group • Responsible for quality assurance

Continued on next page

Role	*Tasks*
Web Application Designer: Olav Bjørkøy	• Main responsibility for compiling prototypes and designing the web application interface. • Assure that the clients specific interface requirements are met.
Database Designer: Riccardo Ortolan	• Responsible for the code to be correct and documented (classes and namespaces). • Responsible for developing the database for the web application.
Test Manager: Erik Andreas Larsen	• Responsible for the overall test plan. • Execution and approval of tests.

2.3 Risk Management

This section present the result from team brainstorming about types of risk linked to this project and how to manage them.

2.3.1 Risk Treatment

The team has agreed upon four ways to handle risk which are taken from the book: Introduction to Risk Management and Insurance (9th Edition). See [1]

- **Avoidance:** elimination of the risk
- **Reduction:** reduce the negative effect of the risk
- **Transfer :** outsourcing or transfer the risk to third party.
- **Retention:** accepting some or all the consequences of a risk

2.3.2 Risk Identification

Explanation of the identified risks:

R1 : Software issues : technical errors and difficulties with git, LaTex and other auxiliary software used for this project which will waste time and/or loss of documents.

R2 : Insufficient knowledge and experience : trying to implement to difficult features in which no value is created for the project.

R3 : Report time consumption : a situation where the team focus to much on the report and falls behind in the implementation part of the project.

R4 : Problems with development model : a situation where the chosen Scrum method proves to have provided too little documentation and planning for the project.

R5 : Ping Pong Lingit : ambiguous Lingit. Changes directive and being diffuse about what they want.

R6 : Fail to meet Lingit's expectations : the risk concerning any misunderstanding about the system requirements.

R7 : Technology issues : the risk that the chosen technology proves to be insufficient for the project.

R8 : Client silence : the team is unable to get in contact with the right contact persons at Lingit, they lose interest in the project.

R9 : Team silence : communication between team members stops.

R10: Supervisor silence : unable to get ahold of the supervisors, or the situation where the team gets too little or no feedback regarding the report and other questions.

R11: Team personnel issues : risk regarding team member sickness like swine flu or regular flu, or group conflicts, member attendance problems or other personnel issues.

R12: Lonesome Cowboy : the situation were a team member goes Lucky Luke and tries to fix or do major changes or implementations to the system/project without the teams approval.

R13: Project recourses : the situation were loss or lack of facilities and equipment hinders the team to implement the project plan.

2.3.3 Risk Assessment

Each risk is evaluated in two dimensions: Probability of occurrence and the extent of damage they will have upon the project if they do occur. Each dimension has three possible levels: low, medium and high. Assessment is done by educated guessing, based upon statistical information and experience. See following page for more details.

Table 2.4: Risk Assessment Table.

Risk	*Description*
ID	R1
Risk Factor	Software issues
Consequence	Medium. Halts the project development by no access to documents.
Probability	Medium
Strategy and actions	Reduce : Need to have proper guidelines on how to use Git properly. One person will specialize in using the software(read: Git and LaTex)
Deadline	Continuous
Responsible	Olav Bjørkøy and Bendik Solheim
ID	R2
Risk Factor	Insufficient knowledge and experience
Consequence	High. The overall success of the project will suffer.
Probability	Medium
Strategy and actions	Reduce : Team members will have to do a lot of self study and provide a good preliminary study to help discover the pitfalls and identify the requirements/features that could prove to be too difficult for the team.
Deadline	Continuous, but more critical in the first sprints.
Responsible	Kjetil Aamodt
ID	R3
Risk Factor	Report time consumption
Consequence	Medium. The team will not have sufficient time to implement the planned features.
Probability	Medium
Strategy and actions	Reduce : Do not reinvent the wheel. Study old reports and use the supervisors active.
Deadline	Continuous
Responsible	Bendik Solheim
ID	R4
Risk Factor	Problems with development model

Continued on next page

Risk	*Description*
Consequence	Medium. Lose the project progress overview. Not knowing what to do next, what we have done and how we should do it.
Probability	Medium.
Strategy and actions	Reduce : Team will have to do Scrum self-study.
Deadline	Sprint 2
Responsible	Erik Andreas Larsen
ID	R5
Risk Factor	Ping Pong Lingit
Consequence	High. The value of previous work will be lowered. Lingit will receive a product based upon our interpretation of their wishes.
Probability	High
Strategy and actions	Reduce : All decisions from meetings must be included in meeting minutes and approved by Lingit.
Deadline	Continuous
Responsible	Brian Mathé
ID	R6
Risk Factor	Fail to meet Lingit's expectations
Consequence	High. Lingit will be unsatisfied and our reputation will lower.
Probability	High
Strategy and actions	Reduce : All decisions from meetings must be included in meeting minutes and approved by Lingit.
Deadline	Continuous
Responsible	Erik Andreas Larsen
ID	R7
Risk Factor	Technology issues
Consequence	High. Previous work will be worthless. A lot of redoing.
Probability	Low
Strategy and actions	Avoid : Good prestudy of the technology we are using.
Deadline	Sprint 1.
Responsible	Riccardo Ortolan and Olav Bjørkøy
ID	R8
Risk Factor	Client silence
Consequence	High. Lingit will receive a product based upon our interpretation of their wishes.
Probability	Medium
Strategy and actions	Reduce: All decisions from meetings must be included in meeting minutes and approved by Lingit.
Deadline	Continuous
Responsible	Brian Mathé

Continued on next page

Risk	*Description*
ID	R9
Risk Factor	Team silence
Consequence	High. Value of created work will be less. Quality of product will drop.
Probability	Medium
Strategy and actions	Reduce : The team will mostly sit and work together, daily scrum meetings.
Deadline	Continuous
Responsible	Everybody has responsibility, main responsible Kjetil Aamodt
ID	R10
Risk Factor	Supervisor silence
Consequence	Medium, reduce the quality of the report.
Probability	Low
Strategy and actions	Accept : Contact the subject teacher.
Deadline	Continuous
Responsible	Riccardo Ortolan
ID	R11
Risk Factor	Team personnel issues
Consequence	High, workload per member will increase. Lower quality and value creation. Bad atmosphere in the group causing a negative working environment negative synergy.
Probability	Medium
Strategy and actions	Accept : Delegate work and re-prioritize the to-do-list.
Deadline	Continuous
Responsible	Kjetil Aamodt
ID	R12
Risk Factor	Lonesome Cowboy
Consequence	Loss of overview/data - a lot of redoing, not following the same project plan, lowering the project/product value and quality.
Probability	High
Strategy and actions	Reduce: Work together most of the time. Close follow-up with daily scrum meeting. Have a concise and updated to-do-list and scope.
Deadline	Continuous
Responsible	Erik Andreas Larsen and Kjetil Aamodt
ID	R13
Risk Factor	Project recourses
Consequence	Work quality goes down. Have to prioritize what work to be done
Probability	Medium

Continued on next page

Risk	*Description*
Strategy and actions	Accept : We have to contact our supervisors and Lingit
Deadline	Continuous
Responsible	Erik Andreas Larsen

2.4 Quality Assurance

This section presents routines and processes to follow to assure the finishing product suitability for its intended purpose and to assure a certain quality of all documents.

2.4.1 Internal Routines

In addition to routines for the client and supervisor interaction, we have developed a set of internal routines to assure a high level of quality throughout the project. The following was agreed upon by all the members of the group at the very first group meeting.

All group members should meet at 08.15 PM Monday through Thursday, and work until 16.00 PM. When subtracting all lectures and meetings, this will give every person roughly 24 working hours each week. This way we are all up to date and available for discussion at least once every day. There will be a very short meeting in the beginning of each day to go through what everyone will be doing that day. This ensures that everyone has a task at all times, and also serves as a way of updating the team members on the the progression so far.

2.4.2 Templates

The group have developed a set of templates for minutes from meeting and agendas, both for the client meetings and the supervisor meetings. These templates can be seen in the appendix for this chapter. In general, the agenda should be short, only serving the purpose of informing the client on the different topics that will be discussed. Minutes from meetings should be detailed enough to include all of the decisions made on the meeting, yet as simple as possible with little to no unnecessary text.

2.4.3 Customer Interaction

To avoid confusion and misunderstandings, some time limits have been set by the project group and agreed upon by the client. These time limits should be followed as far as possible.

Notice of Client Meeting

The project group should warn the client at least 24 hours before the proposed time. This notice should include the agenda for the meeting and any relevant documents that will be discussed at the meeting.

Minutes of Client Meeting

The minutes should be sent to the client within one working day after the meeting. The client should respond and either approve or disapprove the minutes within one working day.

Feedback on phase documents

After each sprint a phase document should be sent to the client as fast as possible. A response should be received by the project group within two working days.

Other Relevant Documents

All documents that require feedback from the client should be sent along with the notice of a meeting in order for them to facilitate what issues will have to be discussed. This should always be a part of the agenda for the meeting.

2.4.4 Supervisor Interaction

There will be a meeting with the two supervisors every Wednesday at 15.00 PM.

Notice of Supervisor Meeting

The agenda and all relevant documents should be sent to the supervisors one working day before the meeting to assure that they have enough time to read them through. See Figure 2.3

Minutes of Supervisor Meeting

Minutes from the meetings should be sent to the supervisors as fast as possible. In case they are not approved, corrections shall be made and sent for approval as fast as possible, at least within 36 hours. See Figure 2.3

Minutes from Client Meetings

The supervisors should also receive the minutes from client meetings.

CHAPTER 3

Preliminary Studies

Preliminary studies aim to give an overview of information needed before we start develop the application. The following elements are covered by the preliminary study:

- **Limitations:** we analyze the limitations of the work in order to get over them.
- **Scrum:** the agile development method we use.
- **Version control:** the tool we use to keep all files up to date while working in parallel.
- **Model-View-Controller:** a discussion about
- **Project risks:** study of the risks we may face during the course of the project.

3.1 Limitations

- **Time:** Everything on the internet takes time to download. Large images, animations and sounds increases the time taken to load a web page, and may cause unnecessary waiting time. We have to be aware that people do not like to wait a lot of time for things, so a few seconds may be enough to lose customers. [10]
- **Browsers:** There are a number of browsers and versions of the same browser. The two widely used are Microsoft Internet Explorer and Mozilla Firefox. They are different and sometimes display pages a bit differently. We will have to pay attention to the layout of the page as stated in table 4.5.
- **Screen resolution:** Most people use a screen of 1024x768 or higher. There is only a minor part of the population that uses a lower one. Some websites say that their page is optimized for a certain resolution. We will not follow this guideline, we are going to make the page work for all or at least most of the screen resolutions [19].

- **Plugins:** Plugins for browsers are small scripts or programs which are made to give additional functionality to the browsers. But which plugins do the visitors have? The problem is that some people do not have them and do not want or do not know how to download them [20]. We are not going to require plugins or external separate programs for visitors.
- **User inexperience:** There is a statistic of how many people do not know how to use the scroll bars. The shortest path to convert a visitor to a customer is to build simple and accessible pages [11].
- **Security:** Security is an important issue on the internet, especially when it comes to services concerning payment. Customers may have virus infected computers, and may therefore upload documents infected with viruses. It may be necessary to scan documents uploaded for viruses [13].

3.2 Scrum

We are using the Scrum method [14] while working on this project. To start using the method we had to find out more about the Scrum since not all of us had worked with the method before and none of us were expert on this subject.

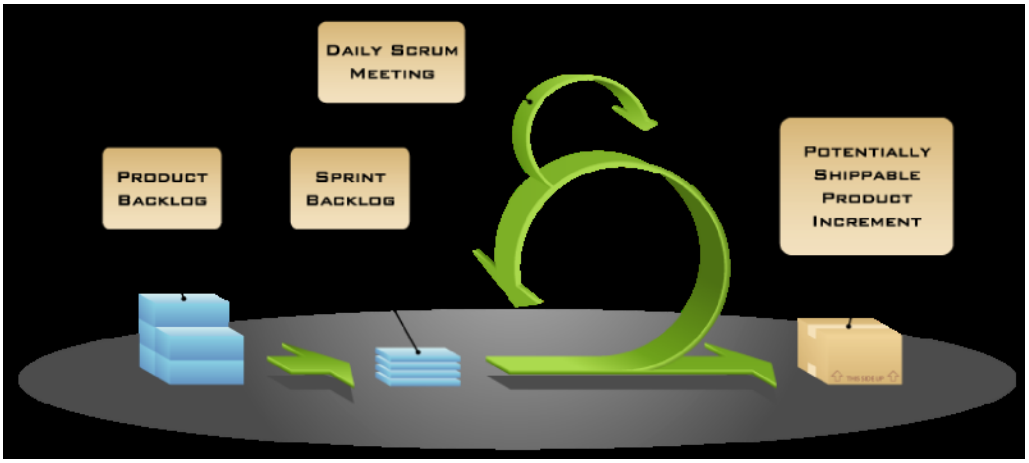


Figure 3.1: Scrum Project Lifecycle

3.2.1 Why Scrum?

The client requested that we would use Scrum instead of the classic waterfall model. Earlier years, the groups have mostly used the waterfall life cycle model [9], but the nature of our task makes Scrum more fitting because the client did not know what exactly they were looking for. Every sprint will add some functionality to the system

and at the end of each sprint we will have a working prototype. We will then at the client meeting show the new features. This will include the client in the development process.

3.2.2 Meetings

Daily Scrum Meetings

We will have one meeting every morning to tell the group what each person is are working on and how the progress is. This a status meeting, also called daily scrum meeting, for the group so we know how far we are along and what each other are doing so we do not do the same work. Since we sit together when we work we can always consult the group with questions.

Scrum Review Meeting

We will have a small presentation in the end of each sprint with the client to show them what have changed since the beginning of the sprint.

3.3 Version Control

We had to choose a version control system before starting to write any code or the report. We have access to a server, and decided on two possible systems, both of which some of the group members had previous experience with.

The first option, SVN, is a version control system with one central repository which resides on the server. Each group member checks out the files, but have to be able to connect to the server to commit new changes.

Git, on the other hand, is a distributed version control system where every group member has a copy of the repository locally, and synchronize their repositories through the server.

We chose to use git, since this ensures that all group members can work on the code and report, regardless of the status of the server. This also means that we have many backups of the entire repository if something goes wrong.

- Git is going to be used as the version controller, so the team can ensure that all the files are up to date and no information is lost during the project.
- It is necessary to work on up-to-date data, especially when working in parallel on source code. When two people are working on the same file they can destroy each others work without knowing it.

- Git supports rapid branching and merging, and includes specific tools for navigating a non-linear development history. A core assumption in Git is that a change will be merged more often than it is written, as it is passed around various reviewers.
- Git also gives each developer a local copy of the entire development history, and changes are copied from one such repository to another. These changes are imported as additional development branches, and can be merged in the same way as a locally developed branch.

3.4 Model - View - Controller

One of the original requirements from the client was that the system would be programmed using Model - View - Controller (hereafter referred to as MVC). MVC is a well-known object oriented programming pattern and is a way to ensure isolation of logic from data and presentation.

3.4.1 How it works

MVC is the concept of separating your code into models, views and controllers. The intention is to make the application more modular to easily be able to change parts of the code without the need for changing code in other modules.

- **Model:** a model is an object that holds data that is required to render information in the view. Depending on the application and the situation, it can be persistent or non-persistent. In the case of web applications it is seldom persistent, it is rather populated each time the object is created.
- **View:** the view is the graphical user interface presented to the user. It renders the information from the model into a readable and understandable form. There is a one-to-many relationship between the model and the view, meaning that one model can have many different views.
- **Controller:** the controller is the link between the model and the view, and contains logic to handle input and output from the view. It does this by making calls to the model and sending the response to the view.

3.4.2 Why MVC?

As already stated, using MVC was more or less a requirement from the client. Nevertheless, MVC would have been an obvious choice irrespectively of the clients preferences. First of all, isolating logic from data and presentation is a smart choice. By doing this

you can alter the way data is displayed without changing the functionality itself, or alter the way data is saved without the need of changing functionality and interfaces as well. All you need to know is how the model, views and controllers are working together. Second of all, MVC has become more or less a de facto standard when developing modern software, both the internet applications and standalone applications. Lastly, using well-known patterns ensures an easier understanding for future developers wanting to further develop the application.

3.5 PHP

PHP is a major topic in our project, and therefore demands some attention before the actual implementation starts. This section includes a discussion around versions, frameworks and some other topics on PHP.

3.5.1 Version

We have chosen to use PHP in order to produce dynamic web pages. It is embedded into HTML and it can be deployed on most web servers and on almost every operating system and platform for free. PHP is free software released under the PHP license, it is installed on over 20 million websites and 1 million web servers [21]. Any PHP code in a requested file is executed by the PHP runtime to create dynamic web page content, and it can be used with many relational database management systems.

From version 5, object handling was completely rewritten, expanding the feature set and enhancing performance. In previous versions of PHP, objects were handled like primitive types[12]. We are using the latest stable version of PHP, version 5.3.0, updated on 2009-06-30. There is also a new version of PHP, 6.0.0, but it is still in development, and we can experience a lot of "beta version" problems using that.

The only big new modifies applied to PHP 6 is unicode support, removal of ereg extension, 'register_globals', 'magic_quotes' and 'safe_mode', and an alternative PHP cache; things that are not going to be considered by us. By using PHP 5.3.0 there is no reason that code for that version will not work on php version 6, at most only a few changes will have to be done[4].

3.5.2 CMS, Framework or Writing From Scratch

One of our most important choices in the planning sprint is whether we should use a PHP framework, a PHP CMS (Content Management System [24]), or write the entire application from scratch.

The client was specific that code quality was paramount to them, and that they expected a well structured final application. They specified that they preferred we use the MVC (Model-View-Controller) pattern to structure our code.

Another important aspect to consider is that very few members of our group has used PHP before. Considering this, it seems to us that the rules and conventions of a framework or CMS would lead to a much more well structured code base, than what the result would be if we wrote the application from a blank slate.

The question then becomes whether a CMS or a framework is best suited for this project. A CMS is most appropriate when the main purpose of the website is publishing information - news, pages, media and other content. A framework, on the other hand, lies on a lower level, and is more suited for websites that function primarily like ordinary applications.

Our project is not about publishing news and other information, but about uploading files, sending emails, administrating customers and employees. We also need a robust API, localization, and other constructs leaning towards using a framework.

Based on this analysis, and the fact that the client wants us to use the MVC pattern (the de facto standard pattern in PHP frameworks), it becomes quite clear that a PHP framework is the best fit for this particular web application.

3.5.3 Framework Requirements

There are too many PHP frameworks to thoroughly try them all out, so we set a few basic requirements to make a framework viable for this project. If we are going to use the framework it needs the following attributes:

1. It should be in active development, so that the client will not be left with an outdated system.
2. The framework should have the main capabilities we need for this site, i.e. sending emails, uploading files, authentication support etc.
3. Since none of us are seasoned PHP developers, and time is of the essence, the framework should have an agreeable learning curve.
4. The framework should be based on the MVC pattern.
5. Built in support for unit testing.
6. The client has specified that the finished system must run on Linux, and that we should use PHP 5.3 and Apache 2. Naturally, the framework of choice should be suited for this environment.
7. The framework should be using a license the client can accept.

3.5.4 Choosing a PHP Framework

Based on the requirements we have for our choice of PHP framework, the following three frameworks seems to be the most applicable to this project:

- Zend Framework (framework.zend.com)
- CakePHP (cakephp.org)
- Symfony (symfony-project.org)

These three are all well known, actively maintained PHP frameworks which meets the requirements outlined above. They can all comfortably run on the client's system, they all use licenses the customer will accept, and they are all capable of using the MVC pattern.

Table 2.3 shows a comparison of the three frameworks based on the framework requirements.

The main point where they differ is on the tradeoff between flexibility and power contra ease of use and learning curve.

Zend Framework is very flexible, and has a modular design which makes it applicable to a wide range of projects. However, as a result of this, it seems to have the steepest learning curve of the three[25].

CakePHP is perhaps the least powerful framework of these three. On the other hand, it has good documentation and seems to be the easiest system to understand and learn. Its learning curve is certainly the most agreeable of the three frameworks[6].

Symfony is not as flexible as Zend Framework, but it is still very powerful[16]. At the same time it has a steeper learning curve than CakePHP. It then fits in between the other two framework in regards to both powerfulness and ease of use.

Since we would like to use the framework with the most agreeable learning curve, the question becomes whether CakePHP is powerful enough for this project. The following list of features would seem to support this:

- Works on PHP 4 and 5
- ORM system for simplified database use
- Request dispatcher for custom URLs
- AJAX helpers
- Built in form validation
- Access control lists
- Data sanitization
- Internationalization and localization

Table 3.1: PHP Frameworks Comparison

Requirement	*ZendFramework*	*CakePHP*	*Symfony*
1. In active development?	Yes	Yes	Yes
2. Has required functionality?	Yes	Yes	Yes
3. Learning curve	Steep	Friendly	Medium
4. Based on MVC pattern?	Yes	Yes	Yes
5. Built in unit testing?	Yes	Yes	Yes
6. Runs on required system?	Yes	Yes	Yes
7. License	BSD	MIT	MIT

- Security, session and request handling components
- Email sending support
- Built in support for unit testing

Based on these features, and further studies of the framework's online documentation, it would seem that CakePHP is more than powerful enough for this project. Since it also seems to be the easiest system to get a great start in, **CakePHP** is our choice of framework.

3.5.5 Model View Controller in CakePHP

Definition

The client requested that the finished product is based on the Model-View-Controller (MVC) architecture, which led to the choice of CakePHP as the underlying framework. In CakePHP MVC means the following:

Model: Each database table has a model, and the model handles all communication with its table.

View: Each page the client can access is a view file, which renders the HTML that gets sent to the Clients browser. Each method in a controller has its own view file.

Controller: The controllers are the glue between models and views. Controllers collect needed data from the models, and sets the variables the views need. Each controller corresponds to a section of the page, and each method in a controller corresponds to an actual page, as rendered by the view file. For instance, the method "about" in the "pages"-controller can be accessed in the browser through the URL "http://example.com/pages/about". The system now has the

By using this architecture, the views never know the structure or operations of the database, and the models never concern themselves with the final presentation in the

views.

Directory structure

The models, views and controllers are organized in the following manner:

```
\app
  \controllers
    name_controller.php
  \models
    model_name.php
  \views
    \layouts
      default.ctp
    \controller_name
      method_name.ctp
```

- Each controller corresponds to a section of the website, for example "customers", "documents" and "admins"
- Each method in a controller corresponds to an actual page on the website, with its own view file. For example, the main home page has the method PagesController:index(), and uses the view file "app/views/pages/index.ctp".
- The view folder also holds "layouts", which are view files that gets wrapped around all other view files.

Routing

This is the way CakePHP routes requests when using MVC:

1. The customer sends a request via the browser, which gets picked up by the Dispatcher.
2. The Dispatcher sends the request to the appropriate Controller, based on the URL.
3. The Controller communicates asks the models for data from the database.
4. The models send data back to the controller.
5. The Controller sets variables for and renders the appropriate View files.
6. The rendered View files (HTML) is sent back to the customer's browser.

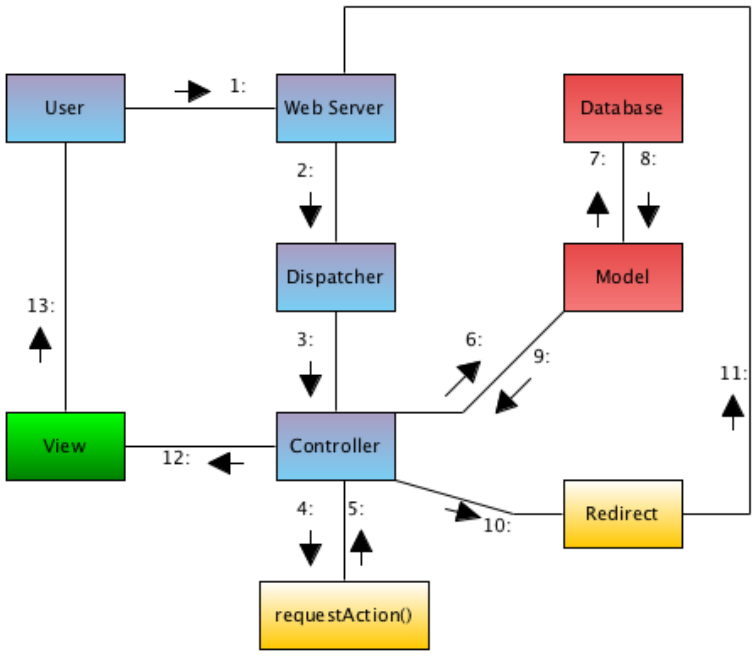


Figure 3.2: CakePHP MVC Request [8]

3.6 Code Conventions

When more than one person work on a coding project, it is important to have a common code convention for everyone to follow. This way it will be easier for all of the involved programmers to read and understand how the code works.

3.6.1 Naming conventions

CakePHP, which is the framework of our choice, already states some naming conventions for files, classes and the database that decreases the amount of configuration needed [7]. Please note that although the term CamelCase can mean both capitalization and de capitalization of the first letter, it will always mean capitalization in this document unless specified otherwise.

File names should always be in lowercase and use underscores instead of whitespace. The filename itself should reflect the class it contains.
Model classnames should be written using CamelCase, and be singular.
Table names corresponding to the models should be plural, lowercase and underscored.
Controller classnames should be written using CamelCase, and be plural. Should always end in 'Controller'.
View template files should be written in lowercase and be underscored. They should

reflect the name of the controller function they display.
Variables should have short, yet meaningful, names. They should be all lowercase without any white spaces or underscores.
Methods should be written using CamelCase, although with the first letter de capitalized.

Using these conventions, you would for example end up having a view called Person. The database table would then be called 'people', and the controller would be called 'PeopleController'. The way this is made up, a request to http://example.com/people/function would map to a call on a function called 'function()' in the controller 'PeopleController'.

CHAPTER 4

Requirements

4.1 Product backlog

The product backlog is all the functional requirements for the system. The backlog is prioritized into three categories: high (H), medium (M) and Low (L). The items categorized as "High" are the ones we have to implement to have a minimally functional web application and text editor add-in. By also implementing the "Medium" requirements, the site and add-on will work about as good as the client wanted. The "Low" prioritized items are features that are nice to have, but that will only be implemented if we still have time after completing the more important items.

This backlog is split into four tables - one for each priority level of the web application, and one for all the items regarding the text editor add-on.

This system will have three kinds of users, which is referred to in the following tables. The types of users are:

- **Customer**: the standard customer who uses the service to translate text snippets and entire documents.
- **Employee**: the translators and proofreaders who work at Lingit, and uses the site to acquire new texts to process.
- **Admin**: the administrators at Lingit who uses the site to administer customers, employees and other administrators.

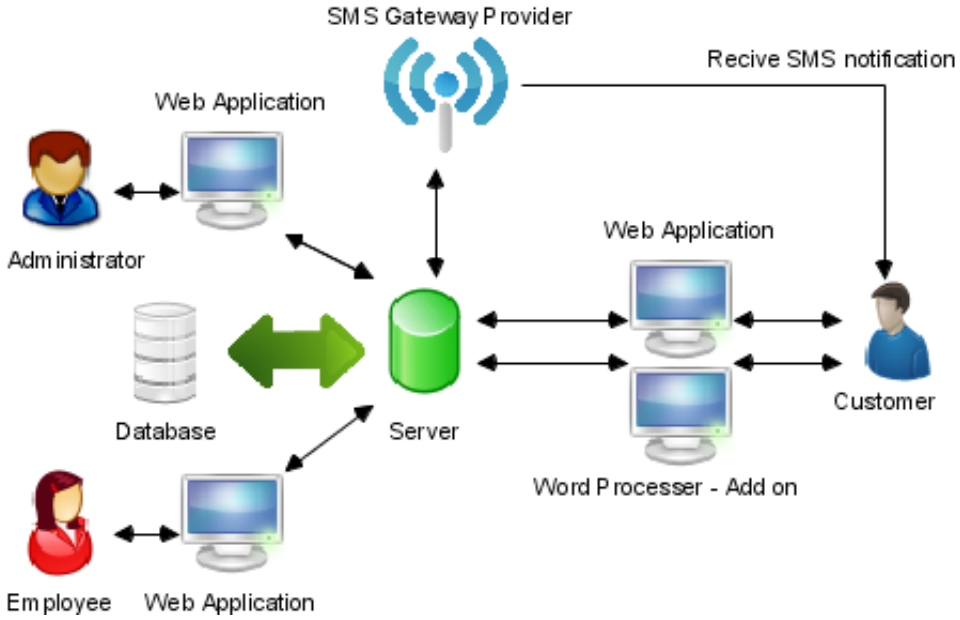


Figure 4.1: High Level System Diagram

4.1.1 Web Application Backlog

The backlog is split into 3 smaller sections, one for each of the priority levels. Each of the backlog tables are split into Customer, Employee, Administrator and sometimes System. This section shows all of the web application backlog items and explains them in detail. The product backlog is a list of tasks we have to complete before the product is complete, and is in many ways the same as functional requirements.

Web Application - High priority items

Descriptions of the items in table 4.1:

Item 1.1 and 1.2: To use the system the customer has to register first and then later log in. This is needed because each customer will have their own account with separate documents and settings.

Item 1.3 and 1.4: Since the idea for this system is to let the customers upload texts for proofreading/translation from a webpage, we have to make some kind of system where the user can select a document that he/she wrote and upload it to the server. For now it will only be able to upload documents of certain types(doc, docx and plain text).

Item 1.5: When uploading, the customer has to specify both the source and target language, so that the system can match the task with a properly skilled employee.

Item 1.6: The credit system is a system to pay for the proofreading/translation. The response time decides how much credit a certain document will cost. The idea is that one word is one credit for the slowest response time and the cost will be the word count multiplied by some number depending on what response time that are chosen.

Table 4.1: Web Application Backlog - High Priority Items

Number	*Priority*	*Task*
Customers		
1.1	H	A customer should be able to register on the website with an email, a password, full name and billing details.
1.2	H	A customer should be able to log in with the username and password after registration.
1.3	H	A customer should be able to paste text into a text box and send it in for proofreading or translation.
1.4	H	A customer should be able to upload Word documents and plain text files to the server for proofreading or translation.
1.5	H	A customer should have the option to choose the language of the text, and the target language if this task is a translation when uploading a document.
1.6	H	A customer should be able to buy more word credits, which are needed to get a text processed, when logged in.
1.7	H	A customer should be able to download the resulting proofread or translated documents when logged in.
1.12	H	When a customer buys more credits, this should be entered in the billing database.
Employees		
1.8	H	An employee should be able to log in if the person is already registered in the system.
1.9	H	An employee should be able to see assigned jobs and possible jobs he or she can take, after logging in.
1.10	H	An employee should be able to download texts assigned to him or her.
1.11	H	An employee should be able to upload finished results and connect these to the original document.
Administrator		
2.18	H	An administrator should be able to see all sent and unsent bills in the database when logged in.

Item 1.7: When logged in, the customer should be able to download the documents that Lingit has finished proofreading or translating. All the previous documents will at all times be saved on the server so the customer can access them.

Item 1.9: After logging in, all tasks suited for this employee are shown by the system.

It also shows those tasks already assigned to the employee.

Item 2.18: When the customer buys more credits the bill will show up at the administrator page. From there the administrator can mark the bill as sent.

Web Application - Medium priority items

Table 4.2: Web Application Backlog - Medium Priority Items

Number	*Priority*	*Task*
Customers		
2.1	M	A customer should be able to upload OpenOffice and PDF documents for translation and proofreading.
2.2	M	A customer should have the option to select a classification of the uploaded text, to further guide the employee in processing the document.
2.3	M	A customer should be able to select if he or she wants finished documents emailed back to their address.
2.4	M	A customer should be able to see the status of any document he or she have previously submitted, when logged in.
2.5	M	A customer should be able to choose the deadline for an uploaded text.
2.6	M	Each page the customer sees should be available in languages the web application supports.
2.17	M	A customer should be able to see pricing options on the home page, without login in.
Employees		
2.7	M	An employee should be able to set his or her availability when logged in.
2.8	M	An employee should be able to set his or her skill sets in different languages, when logged in.
2.9	M	An employee should be assigned texts automatically by the system, based on the employees availability and skill set.
2.19	M	An employee should receive e-mail about new available tasks.
Administrators		
2.10	M	An administrator should be able to log in if registered in the database.
2.11	M	An administrator should be able to see, edit and delete customers, employees and other administrators, when logged in.
2.12	M	An administrator should have the possibility to send emails to employees and customers, when logged in.
2.13	M	An administrator should be able to see the status for all pending jobs when logged in.

2.14	M	An administrator should be able to see all the previous documents and finished texts when logged in.
2.16	M	An administrator should be able to see a log of all recently performed tasks on the website.
System		
5.1	M	The system should report to the administrators if the available employees do not cover supported languages.

Description of the items in table 4.2:

Item 2.1: While plain text (.txt) and MS Office Word formats (.doc, .docx) are the most important for the web application to support, the customer should also be able to upload OpenOffice Writer (.odt) and PDF documents for processing.

Item 2.2 and 2.5: When a customer uploads texts he will have the option to set the classification for the document. This is to better assign a translator for the job since there might be a translator which have more experience in a certain field of work(like legal documents etc.). The customer will also set what language he wants it translated to and from. Proofreading could be seen as a special case of translation, where the original language and translation language is the same. The customer can also choose if he wants an SMS notification and set the deadline. The last choice will change the price of the job.

Item 2.3: The idea is to let the customer decide if the finished result should be e-mailed as a response when finished. It will always be available on the web interface.

Item 2.4: The customer should through the account be informed about the status of any document that is sent in. This will be for example: not assigned, assigned, in progress, finished.

Item 2.5: There are different types of deadlines, which each costs a different amount of credits.

Item 2.6: The customer should be able to select between the following languages on the pages he or she can see: Norwegian bokmål, Norwegian nynorsk, English and German.

Item 2.7 and 2.8: For assigning the jobs better among the different translators they have to set their skill and availability. They have to be able to set which languages they know and how good they are in each. Like a rating. They also need to set when they are available. They might go on some vacation and the system need to know this so it does not assign work for translators that are gone.

Item 2.9 and 2.19: When a new job comes in the system should search for available translator and give the job to the most suiting. If there are two translators available the job should go to the one with the highest rating. The choosen employee should recive an email which notifies the employee about the new job.

Item 2.10 to 2.14: These are the basic tasks performed by an administrator, which in most cases means manipulating the database in some way. This includes the information on all customers, translators and other administrators. This also includes keeping an eye on the documents database table, so that the administrators always know what is going on in the system.

Item 2.16: The administrator will see all actions done. Everyone logging in and out, uploaded documents, user registering, employee responding to a customer.

Web Application - Low Priority Items

Table 4.3: Web Application Backlog - Low Priority Items

Item	*Priority*	*Task*
Customers		
3.1	L	A customer should be able to contact the administrators through the website.
3.2	L	A customer should have the option to get an SMS notification when a translation or proofreading task is finished.
3.3	L	A customer should be able to choose if he or she wants notification by email when a task is finished.
Administrators		
3.4	L	An administrator should be able to add new administrators to the system when logged in.
3.5	L	An administrator should be able to add new employees to the system when logged in.

Descriptions of the item in table 4.3:

Item 3.1: When logged in it should be a form where the customer can easy send questions to Lingit. It will be the administrator that will delegate the question further is he/she can not answer them.

Item 3.2: Since the customer can choose to get a SMS notification when the document is finished, we need to make a system that does that. This will be done by connecting with an existing third party company and let them take care of the sending of SMS.

Item 3.3: A customer should be able to turn the email notifications on or of through the website.

Item 3.4: Administrators will be able to add users with the same system rights as himself(administrator rights).

Item 3.5: Administrators will be able to register new employees in the system.

4.1.2 Add-on backlog

Table 4.4: Add-on Backlog

Item	*Priority*	*Task*
4.2	H	The customer should be able to select text in the editor and press Proofread or Translate.
4.3	H	The add-in should ask the user for username and password, job deadline, text type, the source language and optionally the target language.
4.4	H	The add-in should send the selected text to the server API, together with the user details.
4.5	H	The web application should send an email to the user when the translation from the add-in is finished.
4.6	M	The add-in should be able to check the server for results.
4.7	M	The add-in should download the resulting text and show it in a window.
4.8	M	The customer should be able to download an installer file to install the MS Word macro add-in.
4.9	L	The MS Word macro add-in should be ported to C#.
4.9.1	H	The add-in should detect if the user is connected to the internet.
4.9.2	M	The add-in should have a button for checking for news. When there are new documents, there should be a link to the web application displayed when the user presses that button.
4.9.3	M	The add-in should remember the username and the password of the customer in the current session.
4.9.4	M	The add-in should remember the username of the customer when MS Word is closed and then started again.
4.9.5	M	The add-in should be able to check for news on startup.
4.9.6	M	The add-in should automatically set the default language of the language from field to the language being used in the document being written.
4.9.7	M	The add-in should automatically get the available languages from the server.
4.10	L	The add-in should be ported to work in the latest version of OpenOffice Writer.

Descriptions of the item in table 4.4:

Item 4.2: In MS Word the user can select text and then choose to send it to the server just as if he uploaded it through the web interface.

Item 4.3: When the user does this he should be asked for a password and user name. This should be a pop-up box in Word where the user can enter the password and user name.

Item 4.4: When the user press send on the pop-up box the selected text will be made into a xml-string and sent to the API with an http post request.

Item 4.5: As it is when a customer uploads a document on the website, the system sends an email when the task is finished. This provides a simple backup of the result for the customer.

Item 4.6 and 4.7: When the user opens Word the add-in should automatically query the server to see if there are any newly finished documents. If there is, the user should be presented with a link to the web application.

Item 4.9: If time allows us to, we will port the add-in to the latest version of OpenOffice Writer, so that more users can be supported by the service.

Item 4.10: Lingit requested that we should port the MS Word add-in from VBA to the C# programming language, if we have available time remaining at the end of the project.

4.2 Non-functional Requirements

Item 1.1, 2.1, 2.2, 2.3: Technical details from the client, because this is the system they will be installing the application on. This is therefore fundamental for the project as a whole, as the implementation in the end depends on this.

Item 1.2, 1.3: A special request from Lingit. Using MVC makes it easier to change only the underlying data model of the system. It is also especially useful in web based applications, where the view itself often is on a different location the the model and controller. The MVC architecture also fits very well with the object oriented architecture.

Item 2.4: The layout engines listed are the engines the application will be tested in.

Item 2.5: To make it as easy as possible for the client to implement the system, this requirement specifies that a document explaining every aspect of the implementation must be delivered with the system itself.

Item 2.6: Because this is a 'small' project in context to real world projects, it is important to document the code well enough for the client to be able to develop it further. This includes commenting every part of the code, as well as following all conventions decided by the group.

Item 3.1: A person using a 'modern' computer has to be able to use this service at all times, excluding scheduled and announced downtime.

Item 3.2, 3.3: The system serves as a small but important service to customers, and should therefore not be something a paying customer must spend an exceeding amount of time to understand and use. There will also be a range of different

customers with different knowledge of information systems who will be using this system, and everyone should be able to us it without further problems.

Item 4.1: Because this is for non advanced users it should be easy to install the add-in on their system.

Item 4.2: The code should not be available for everyone. This is because we do want people to see the code and use it in their own system.

Item 4.3: The install executable will work on Microsoft Office Word 2007. Earlier version of Microsoft Office Word are not considered.

Table 4.5: Non-Functional Requirements

Number	*Task*
Language	
1.1	The application should be written in HTML 4.1 and PHP 5.3.0.
1.2	The web application should be based on the MVC architecture.
1.3	The web application should be written using an object-oriented architecture.
Standards and implementation	
2.1	The application should use MySQL 5.1 as the only database.
2.2	The application should be possible to implement on web servers running Apache 2.X or newer.
2.3	The application should run on a Linux server.
2.4	The application should work as intended for browsers using the following layout engines: IE6+, Gecko 1.9+, Presto 2.0+ and WebKit 522+.
2.5	Implementation documentation and deployment method has to be provided together with the finished system.
2.6	The code should be properly documented for future development.
Reliability and Usability	
3.1	The users should be able to access the site and their accounts 24 hours a day, seven days a week, excluding scheduled downtime and maintenance that the users have been notified about in advance.
3.2	For the application to be fast and responsive, all page request should be answered in less than 0.5 seconds, as long as the customers available bandwidth is not the bottleneck.
3.3	The application should follow established standards for page layout and navigation, and present information in generally readable font sizes and colors.
Add-on	
4.1	It should be able to install the add-on with an executable.
4.2	The code should be protected from everyone to see
4.3	The add-in should work in Microsoft Office Word 2007.

4.3 Use Cases

After considering the functionality of the system and discussing how the system should work in general with the customer, we developed some use cases to describe this in an easier manner. Following is an UML representation of the use cases, as well as a short description of each of the tasks. As can be seen from the use cases, the system consists of three different parts; the customer interface, the employee interface and the administrator interface.

Figure 4.2: Overall Use Case Diagram

Figure 4.3: Create Account Use Case

Table 4.6: Create Account Description

Name: UC-00: Create account	**Objectives**: Add a new customer to the database so he or she can use the service.
Primary actors: A customer **Secondary actors**: None	**Dates**: 03.09.09 **Responsible**: Brian Mathé **Version**: Sprint 1
Preconditions: The system is available for adding a new customer and there must be some place in the database for him/her.	**Postconditions**: The system is added a new customer in the database.
Scenarii: The customer click on the "Register" button. He or she is then directed to a new page where he or she can fill in a form with all the informations needed. When clicking OK, the system should first verify if the type of informations are right. If they are, the informations are added to the database. If they are not, the same page is reloaded on the customer's browser with warnings about what are the wrong informations.	

Figure 4.4: Customer Login Use Case

Table 4.7: Customer Login Use Case Description

Name: UC-01: Customer log in	**Objectives**: Connect a customer to the system so he or she can access to his or her informations
Primary actors: A customer **Secondary actors**: None	**Dates**: 03.09.09 **Responsible**: Brian Mathé **Version**: Sprint 1

Preconditions: The customer is already registered (in the database) and not connected.	**Postconditions**: The customer is connected and can see all his/her details. He then can start to use all the functions of the system.
Scenarii: The system should send a query to the database in order to get all the personal details of the customer. That means the system get the name and last name, the names, type (Word, OpenOffice, plain text...) and status of the documents, the amount of words available on his or her account and show these informations on the customer's personal page.	

Figure 4.5: Previous Documents Use Case

Table 4.8: Previous Documents Use Case Description

Name: UC-02: See previous documents	**Objectives**: A customer should be able to see the progress of his or her documents.
Primary actors: A customer **Secondary actors**: None	**Dates**: 03.09.09 **Responsible**: Brian Mathé **Version**: Sprint 1
Preconditions: The customer is logged in (status: active)	**Postconditions**: None.
Scenarii: Wether the customer is trying to see a Word, OpenOffice or plain text file, the click on one of his or her document will start a download in the browser.	

Figure 4.6: Documents in Progress Use Case

Table 4.9: Document in Progress Use Case Description

Name: UC-03: See document progress.	**Objectives**: A customer can be able to see the progress of his or her documents.
Primary actors: A customer **Secondary actors**: None	**Dates**: 03.09.09 **Responsible**: Brian Mathé **Version**: Sprint 1
Preconditions: The customer is logged in.	**Postconditions**: None.
Scenarii: The customer, on his home page, can see the status of his or her documents. That includes 4 different statuses: not assigned, assigned, in progress and finished.	

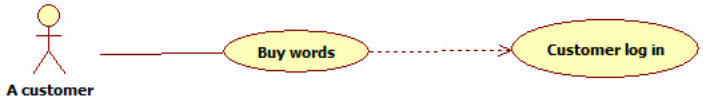


Figure 4.7: Buy More Words Use Case

Table 4.10: Buy More Words Use Case Description

Name: UC-03: Buy words.	**Objectives**: The customer can be able to buy new words for his or her forward documents.
Primary actors: A customer **Secondary actors**: None	**Dates**: 03.09.09 **Responsible**: Brian Mathé **Version**: Sprint 1
Preconditions: The customer is logged in.	**Postconditions**: Money has been made !

Scenarii: The customer click on a "Buy new words" button and chooses how much words he wants to purchase. This amount of words is then subtracted from his account. The system should also send to a predefined printer at Lingit a pre-formatted bill that will be taken by the secretariat and sent to the postal address of the customer. The customer should then pay the bill.

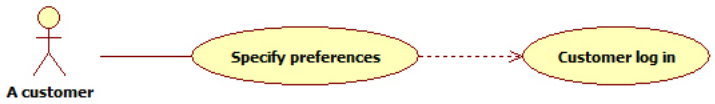


Figure 4.8: Specify Preferences Use Case

Table 4.11: Specify Preferences Use Case Description

Name: UC-04: Specify preferences.	**Objectives**: The customer should be able to specify the his or her e-mail & password. Remark: the language of the system can be changed on clicking on the flag of the language wanted.
Primary actors: A customer **Secondary actors**: None	**Dates**: 03.09.09 **Responsible**: Brian Mathé **Version**: Sprint 1
Preconditions: The customer is logged in.	**Postconditions**: The informations about the customers are updated.
Scenarii: Same thing as "See previous documents" use case but the way it will be displayed will change. New documents will appear in the "new documents section" on the customer's home page.Once on his or her homepage, the customer click on a link to a page where he or she can set up his or her preferences. The system has then to load a new page with the preferences of the customer with a query to the database. He or she can set up there the language of his or her homepage. When the click on "Save preferences" button is pushed, the new informations are uploaded in the database.	

Figure 4.9: Document Proofread Use Case

Table 4.12: Document Proofread Use Case Description

Name: UC-05: Submit new task	**Objectives**: Submit a new document to be translated and/or proofread by a employee.
Primary actors: A customer **Secondary actors**: An employee	**Dates**: 03.09.09 **Responsible**: Brian Mathé **Version**: Sprint 1
Preconditions: The customer is logged in.	**Postconditions**: A new document has been added to the database, the status of the customer's document has been modified and an e-mail has been sent to the employee.
Scenarii: From his or her homepage, the customer click on a "add a new document" button. When click is done, the system loads a page in which the customer can either paste a plain text in the dedicated form or upload a file from his or her computer. He or she has to set up the following details: if he or she wants an SMS notification or not, the language of the documents he or she is uploading and the one in which he or she wants the document translated to, the response time, a possibility to add the field of study of the document from a drop down menu. When this is done, a click on the "Send document" button is sent to the database. See "Input text" and "Upload document" for more details for each scenario. The system should then look if a field of study is specified. If so, it should search for an available skilled employee for this task and assign this task to him or her by sending an e-mail. Finally it should modify the status of this document to "assigned". If no employee is or no skilled employee for this task are available, the system should set the status of the document to "not assigned".	

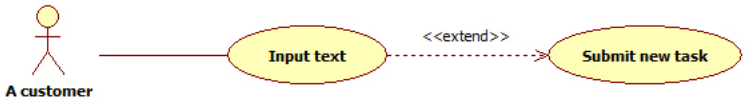


Figure 4.10: Input Text Use Case

Table 4.13: Input Text Use Case Description

Name: UC-06: Input text	**Objectives**: Submit a new document in pasting a text.
Primary actors: A customer **Secondary actors**: An employee	**Dates**: 03.09.09 **Responsible**: Brian Mathé **Version**: Sprint 1
Preconditions: The customer is logged in.	**Postconditions**: A file has been created and added to the database, the status of the customer's document has been modified and an e-mail has been sent to the employee.
Scenarii: This is a sub-case of the previous Use case. While pasting his or her text to be treated by the employee, the customer should see the number of words he or she used with this text appear in real-time under the input box. This should be made with a script in javascript. When clicking on the "Send document" button, the system should create a .txt file and store it in the database.	

Figure 4.11: Upload Job Use Case

Table 4.14: Upload Job Use Case Description

Name: UC-07: Upload new job	**Objectives**: Submit a document in uploading a file.
Primary actors: A customer **Secondary actors**: An employee	**Dates**: 03.09.09 **Responsible**: Brian Mathé **Version**: Sprint 1

Preconditions: The customer is logged in.	**Postconditions**: A file has been added to the database, the status of the customer's document has been modified and an e-mail has been sent to the employee.
Scenarii: The customer should click a button where he or she can upload the file from his or her computer. When the fils is selected, an upload to the database in being processed by the system. When finished downloading, a script implemented in PHP should be able to count the word in the file. We assume that this script can do this with the following file extensions: .doc, .docx and .odt. The number of words in the documents should then be displayed in the browser of the customer.	

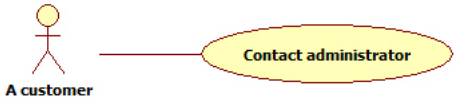


Figure 4.12: Contact Administrator Use Case

Table 4.15: Contact Administrator Use Case Description

Name: UC-08: Contact administrator	**Objectives**: A customer should be able to contact an administrator
Primary actors: A customer **Secondary actors**: An administrator	**Dates**: 03.09.09 **Responsible**: Brian Mathé **Version**: Sprint 1
Preconditions: The customer is logged in.	**Postconditions**: None
Scenarii: A link on the website should be displayed. When clicking this link, the customer should be able to send an e-mail to one of the administrators of the system.	

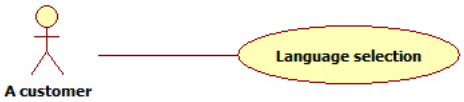


Figure 4.13: Language Selection Use Case

Table 4.16: Language Selection Use Case Description

Name: UC-09: Language selection	**Objectives**:A customer should be able to select the language he or she wants for the website.
Primary actors: A customer **Secondary actors**: An administrator	**Dates**: 03.09.09 **Responsible**: Brian Mathé **Version**: Sprint 1
Preconditions: None	**Postconditions**: None
Scenarii: Some flags corresponding of the languages available for the system should be displayed. When clicking on one of those flags, the website should change the language of the texts.	

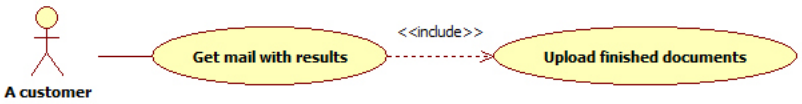


Figure 4.14: Mail Notification Use Case

Table 4.17: Mail Notification Use Case Description

Name: UC-10: Get mail with results	**Objectives**: The customer is notified by e-mail when the job is done.
Primary actors: A customer **Secondary actors**: None	**Dates**: 07.09.09 **Responsible**: Brian Mathé **Version**: Sprint 1
Preconditions: The status of the document is achieved	**Postconditions**: The customer has received an e-mail.
Scenarii: When the employee uploads an achieved file on the server (status:achieved), the system should look at the e-mail address of the customer and therefore send him or her an e-mail notification. The e-mail should contain the document expected be the customer in an attached file.	

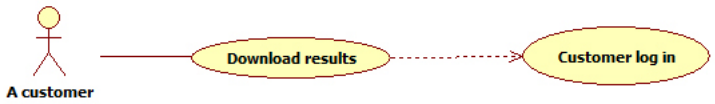


Figure 4.15: Download Results Use Case

Table 4.18: Download Results Use Case Description

Name: UC-11: Download Results	**Objectives**: The customer can download the finished document when the job is done.
Primary actors: A customer **Secondary actors**: None	**Dates**: 07.09.09 **Responsible**: Brian Mathé **Version**: Sprint 1
Preconditions: The status of the document is achieved	**Postconditions**: The user get the document, but no changes to the system
Scenarii: When the employee uploads an achieved file on the server (status:achieved), the document should be able for downloading at the customer page.	

Figure 4.16: SMS Notification Use Case

Table 4.19: SMS Notification Use Case Description

Name: UC-12: Receive SMS notification	**Objectives**:The customer is notified on his or her cell phone when a job is done by an employee.
Primary actors: A customer **Secondary actors**: None	**Dates**: 07.09.09 **Responsible**: Brian Mathé **Version**: Sprint 1

Preconditions: The status of the document is achieved and the customer chose to receive an SMS notification.	**Postconditions**: The customer has received an SMS and his account has been subtracted the amount of the SMS.
Scenarii: When the employee uploads an achieved file on the server (status:achieved), the system should see if there is an SMS notification wanted with this file. If so, the system should look at the person the document belongs to and then send SMS to the customer number.	

Figure 4.17: Employee Login Use Case

Table 4.20: Employee Login Use Case Description

Name: UC-13: Employee log in	**Objectives**:Log in the employee
Primary actors: An employee **Secondary actors**: An administrator	**Dates**: 07.09.09 **Responsible**: Brian Mathé **Version**: Sprint 1
Preconditions: The employee has been registered by an admin.	**Postconditions**: The status of the employee has been changed.
Scenarii:The employee opens the system homepage, click a button, enter his login and password, click on the "connect" button. His or her status is then changed to "active".	

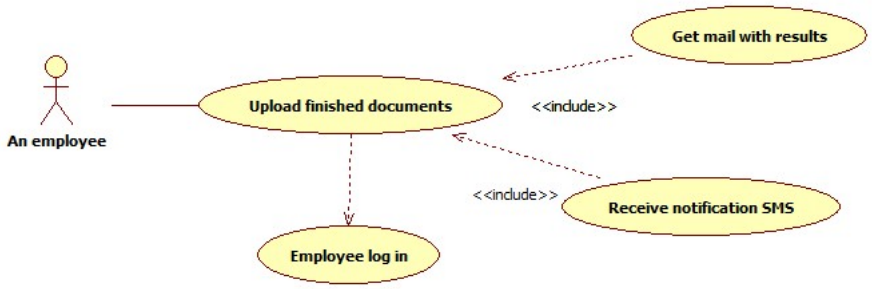


Figure 4.18: Upload Finished Document Use Case

Table 4.21: Upload Finished Document Use Case Description

Name: UC-14: Upload finished documents.	**Objectives**:Upload to the system a finished document so that the customer can see it quickly when it is done.
Primary actors: An employee **Secondary actors**: A customer	**Dates**: 07.09.09 **Responsible**: Brian Mathé **Version**: Sprint 1
Preconditions: The employee has been registered by an admin.	**Postconditions**: A new document had been added to the database and the status of the corresponding task has been changed to "achieved".
Scenarii: As soon as the employee has finished his or her translation or proofreading, the employee should click on a link on his or her personal page allowing him or her to upload the file corresponding to a given document. When the upload is finished, the status of the document is automatically changed to "achieved".	

Figure 4.19: Take Jobs Use Case

Table 4.22: Take Jobs Use Case Description

Name: UC-15: Take jobs	**Objectives**:Assign a task to an employee
Primary actors: An employee **Secondary actors**: A customer	**Dates**: 07.09.09 **Responsible**: Brian Mathé **Version**: Sprint 1
Preconditions: A customer has uploaded a new task.	**Postconditions**: The status of the document has been changed and the customer is notified that his document is taken care of.
Scenarii: The employee should, in priority, take the tasks previously assigned to him or her. If no special task is assigned to him, he or she should have a look at the other ones. When decided on what task he or she has decided to take, he or she should click on a button "Take job". This will automatically change the status of the document to "in progress". This way, the customer can see that his or her task has is being treated at the moment.	

Figure 4.20: Download Documents Use Case

Table 4.23: Download Document Use Case Description

Name: UC-16: Download document	**Objectives**:Able an employee to download a document.
Primary actors: An employee **Secondary actors**: None	**Dates**: 07.09.09 **Responsible**: Brian Mathé **Version**: Sprint 1
Preconditions: The status of the document is "in progress".	**Postconditions**: None.

Scenarii: After having taken the task (see previous Use case description), the employee is offered to download the document. Though, he or she can decide to treat this task later in downloading the file from his or her personal homepage.

Figure 4.21: Receive Mail About New Tasks Use Case

Table 4.24: Receive Mail About New Tasks Use Case Description

Name: UC-17: Get mail about new tasks	**Objectives**:Warn the employee by e-mail.
Primary actors: An employee **Secondary actors**: None	**Dates**: 07.09.09 **Responsible**: Brian Mathé **Version**: Sprint 1
Preconditions: The employee has been assigned a document.	**Postconditions**: None.
Scenarii: When a task has been submitted to an employee, the system should automatically send him or her an e-mail.	

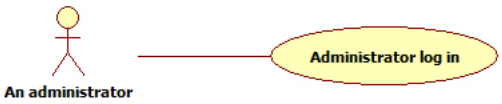


Figure 4.22: Administrator Login Use Case

Table 4.25: Administrator Login Use Case Description

Name: UC-18: Administrator login	**Objectives**:Log in an administrator.
Primary actors: An administrator **Secondary actors**: None	**Dates**: 07.09.09 **Responsible**: Brian Mathé **Version**: Sprint 1

Preconditions: None	**Postconditions**: The administrator is logged in
Scenarii: Same as customer and employee log in.	

Figure 4.23: Add New Employee Use Case

Table 4.26: Add New Employee Use Case Description

Name: UC-19: Add new employee	**Objectives**:Add new employee to handle the tasks.
Primary actors: An administrator **Secondary actors**: An employee	**Dates**: 07.09.09 **Responsible**: Brian Mathé **Version**: Sprint 1
Preconditions: The administrator is logged in.	**Postconditions**: New employee in the database.
Scenarii: Once logged in on his or her homepage, an administrator should be able to click on an "add employee" button and then be directed to a page in which he or she can set up all the details about the employee he or she is adding. When clicking on the "submit" button, a new employee is added to the database a notification by e-mail is sent to him.	

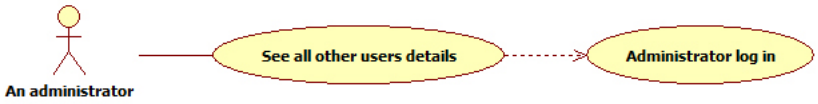


Figure 4.24: See All User Details Use Case

Table 4.27: See All User Details Use Case Description

Name: UC-20: See all users details	**Objectives**:See all users details.
Primary actors: An administrator **Secondary actors**: None	**Dates**: 07.09.09 **Responsible**: Brian Mathé **Version**: Sprint 1
Preconditions: The administrator is logged in.	**Postconditions**:None
Scenarii: On his or her homepage, the administrator should be able to click a button directing him to the list of users. He or she should be able to sort them out by criterions (name, amount on account, last login...). *Note: User means here employee or customer.*	

Figure 4.25: Delete Users Use Case

Table 4.28: Delete Users Use Case Description

Name: UC-21: Delete users	**Objectives**:Delete users
Primary actors: An administrator **Secondary actors**: None	**Dates**: 07.09.09 **Responsible**: Brian Mathé **Version**: Sprint 1
Preconditions: The administrator is logged in.	**Postconditions**:A user is deleted from the database.
Scenarii: On his or her homepage, the administrator can click on a button "delete user" which will delete the user his or her details from the database.	

Figure 4.26: Add New Administrator Use Case

Table 4.29: Add New Administrator Use Case Description

Name: UC-22: Add new admins	**Objectives**: Add new admins
Primary actors: An administrator **Secondary actors**: Another administrator	**Dates**: 07.09.09 **Responsible**: Brian Mathé **Version**: Sprint 1
Preconditions: The administrator is logged in.	**Postconditions**:A new administrator is added to the database.
Scenarii: Once logged in on his or her homepage, an administrator should be able to click on an "add administrator" button and then be directed to a page in which he or she can set up all the details about the administrator he or she is adding. When clicking on the "submit" button, a new administrator is added to the database a notification by e-mail is sent to him.	

Figure 4.27: Send Mail Use Case

Table 4.30: Send Mail Use Case Description

Name: UC-23: Send e-mail	**Objectives**: The administrator is able to send an e-mail to every user.
Primary actors: An administrator **Secondary actors**: Any other user concerned.	**Dates**: 07.09.09 **Responsible**: Brian Mathé **Version**: Sprint 1

Preconditions: The administrator is logged in.	**Postconditions**:None
Scenarii: When browsing the details of the users, a "Send e-mail" button should be available. When clicking it, the administrator is invited to send an e-mail to the selected user.	

4.4 Test Plan

This chapter describes the test plan for this project. The purpose of this plan is to provide a framework for the testing and to assure that tests are written and executed in an organized fashion. The intention of each test is non-trivial and seeks to ensure that the requirements are implemented in a satisfying manner. Thus, the testing part is very important since it will document that all the parts of the system works and behave properly according to its purpose. Testing will also be a valuable method to keep system errors and behavioral exceptions to a minimum by making it easier to identify them. All test cases are mostly based upon the product backlog, section 4.1, and the use cases, section 4.3.

The test plan mainly follows the structure of IEEE 829-1998 - IEE Standard for Software Test Documentation [5], but with some modifications which has been done to make the test plan better suit this project.

4.4.1 Test Items

Scope of testing will include:

- The Lingtrans Web Application and its infrastructure; Customer, employee and the administrator interface.
- Only items which are specified in the requirements chapter, section 4.1 and 4.2.

Scope of testing will not include:

- The Lingtrans add-on for Microsoft Office; specified in the add-on backlog in the requirements chapter.
- Implementation and code documentation.

4.4.2 Testing Approach

This section describes the overall approach and strategy to testing.

- The test plan document and requirement chapter forms the basis of the testing, and should therefore be read and understood by all the testers.

Organization of tests: The team has decided to use a functional approach (also known as black-box testing, see figure 4.28). We found this the most appropriate way to test when building a web application, listed below.

- Testing can start early.
- Testing can be conducted by other team members than the programmer.
- Continuously testing is possible, which will help discover possible errors; Features are tested more then once.
- The testing will follow the backlog items in a naturally order. (Customer registration function will be completed before Administrator delete function will be completed). Top to bottom completion.

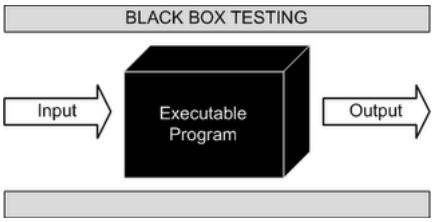


Figure 4.28: Black-box Testing

Conduct tests: by having a selective focus. This means that the team will focus on identifying tests that will uncover the most errors and which are regarded as being the most critical parts of the system. This is necessary to keep the number of tests and test documentation within the projects time limit and resources. Some key questions which will help detecting the critical test objects are listed below.

- Are there any system units that are used more often, or by several other units?
- Is there a function which does complex computations or processes?
- Is there a function that is crucial to meet a business requirement?

A combination of manual and automatic testing will be used:

- Unit testing is to be written and executed by the programmer responsible for the implementation of that unit, using the SimpleTest framework provided in CakePHP.

- System testing is to written and executed by the person resonsible for the spesific part of the web application, using the SimpleTest framework provided in CakePHP.

For each test there will be a test case description and a test case execution log. See table 4.33 and table 4.34, respectively.

4.4.3 Features to be tested:

This section specifies the test design for the features to be tested. The team has identified six categories of tests that the features will be divided into. A short description of each category is listed below.

- **Unit testing:** Tests the most critical functions to assure that they behave correctly according to their intentions.
- **System testing:** Tests of the entire system as a whole, which ensures that the different individual parts works together in context to the system requirements. This implies that most of the system must be implemented before the execution of these tests.
- **Performance testing:** Tests the measurable aspects of the system; how fast the system responds.
- **Usability testing:** Tests how real users interact with the system. This will provide a good indication of how user-friendly the system is.
- **User acceptance test:** This is where the customer tests the final product and approve, or disapprove, the product as an acceptable solution to their given task.

4.4.4 Features not to be tested

- Formal tests of all the units in the system will demand to much documenting; formal tests of non-critical units will not be created in according to the selective focus approach.

4.4.5 Test Identification

The team has created an unique code for every category of test we have described which will be be used to identify each test case. The code consist of four letters describing the test category and a two numbers describing the test number within that particular test category, see table 4.31.

Table 4.31: Test Identification.

Testcategory	*Code*
Unit test	UNTE-xx
System and Web test	SYWE-xx
Performance test	PETE-xx
Usability test	USTE-xx
Acceptance test	ACTE-xx

4.4.6 Test Case Overview

This section presents an overview over all the test cases. A complete description of each test case can be found in appendix A.2.

Unit tests:

- UNTE-00 - Document Controller Function Test
- UNTE-01 - Email Notification Function Test
- UNTE-02 - Framework Controller Tests
- UNTE-03 - Framework Models Tests
- UNTE-04 - Framework Helpers Tests
- UNTE-05 - Framework Database Tests

System tests:

- SYWE-00 - User Registration and Login
- SYWE-01 - Customer Account Settings
- SYWE-02 - Administrator Management
- SYWE-03 - Employee Availability

Performance tests:

- PETE-00 - Performance Test

Usability tests:

- USTE-00 - Customer Usability Test

Acceptance tests:

- ACTE-00 - Client Acceptance Test

4.4.7 Item Pass / Fail Criteria

A item has passed the test if the output of the test equals the anticipated outcome for the given input. A item has failed the test if the output of the test differs from the anticipated output for the given input, or if the pre-condition is not fulfilled.

4.4.8 Test Schedule

Table 4.32: Test Schedule.

TestID	*TestExecution*
UNTE-00	09.10.2009
UNTE-01	23.10.2009
UNTE-02	05.11.2009
UNTE-03	05.11.2009
UNTE-04	05.11.2009
UNTE-05	05.11.2009
SYWE-00	09.10.2009
SYWE-01	09.10.2009
SYWE-02	02.11.2009
SYWE-03	23.10.2009
PETE-00	06.11.2009
USTE-00	06.11.2009
ACTE-00	19.11.2009

4.4.9 Test Case Templates

Test description templates to be used for each test. The purpose of this template is to have a standard and to make sure that adequate information about the test-cases is provided.

Table 4.33: Test Description Template.

Title	*Test Title.*
Test Identifier	Category-identifier + test case number.
Dependencies	Other test which needs to be approved first.
Responsible	Team-member or external person who is responsible for the test.
Feature to be tested	What functionality is to be tested.
Features included	Which other features functionality are also tested or included in the test.
Pre-conditions	Pre-conditions that must be fulfilled in order to execute the test.
Execution steps	What steps that need to be conducted in order to reach the expected result of the test.
Expected result	The expected outcome of the test.

Table 4.34: Test Execution Template.

Title	*Testtitle.*
Test Identifier	Category-identifier + test case number (Links the Test Description and Test execution).
Date	Date when executed
Executioner	Team-member or external person who execute the test.
Test Result	Description of the result of the test.

4.4.10 Environmental Needs

Hardware needs: A modern computer which is able to run the necessary software.

Software needs:

- Web browser: Safari, Mozilla Firefox and Internet Explorer.
- CakePHP SimpleTest framework.

- Apache server with PHP and mySQL.

Personnel needs: All tests, except the usability and customer acceptance tests are conducted by the team members. The former requires potential users of the finished system, and the latter requirers personnel from Lingit.

4.4.11 Responsibilities

The test manager is responsible for writing the test plan and to manage all the formal test cases described in this document. For informal tests; Each programmer is responsible for writing and testing the feature that they are implementing. The test manager is to assure that each test cases happens within the scheduled time, and that the test cases are written and executed correctly.

Part II

Sprints

Introduction

Purpose

The purpose of this part is to provide a detailed step-by-step description of our solution to the task. This part explains the implementation process of the system requirements defined in part I, as they are carried out incrementally in each sprint. The goal is to make the reader understand how the system is built and acquire the knowledge needed to maintain or further develop the system, if needed.

Scope

Each chapter presents one sprint in detail: sprint goal, planning of the sprint, which of the product backlog items we focused on and how they were implemented, test results and an evaluation of the sprint as a whole.

Sprint 1 presents the first wireframes and the underlying database which is the foundation for further sprints. **Sprint 2** focuses on implementing high priority backlog items, such as user login and most of the customer requirements. **Sprint 3** describes how the medium priority items of the system was implemented. Test results of one of the most important features, the document controller, is shown. In addition, we had product backlog and non-functional requirements changes in this sprint, which are also explained. In **Sprint 4** most of the low priority items was implemented, and most of the administrator interface of the web application is finished. The final version of the database is presented. **Sprint 5** presents all the final system tests and the finishing touches of the implementation, which focus on the add-in.

CHAPTER 5

Sprint 1

The first sprint is only a short sprint, lasting one week, where we will focus on predevelopment of the system. In this sprint we will look at three different parts of the system: the database model, an add-on for Microsoft Word and OpenOffice Writer, and the application itself. We will try to come up with a more or less finished database model so we can focus more on the programming in sprint 2. For the add-on, we will be examine what possibilities there are, and how add-ons for Microsoft Office and OpenOffice are normally developed. We will not make any alpha or pre alpha version of the add-on. Finally, we are going to make some wireframes for the web application to show the client what we believe the system should look like in the end. These wireframes will be a visual representation without any functionality, but should also give a good overview for the client.

5.1 Sprint Planning

This section includes the overall goals of this sprint, as well as a more specific sprint backlog.

5.1.1 Overall Sprint Goal

The goal for sprint 1 is to have a working database model, investigate the possibilities around developing an add-on for Microsoft Word and OpenOffice, and have an early version of the web application without any functionality ready. This goal is illustrated in the High Level System Focus Diagram 5.1 below.

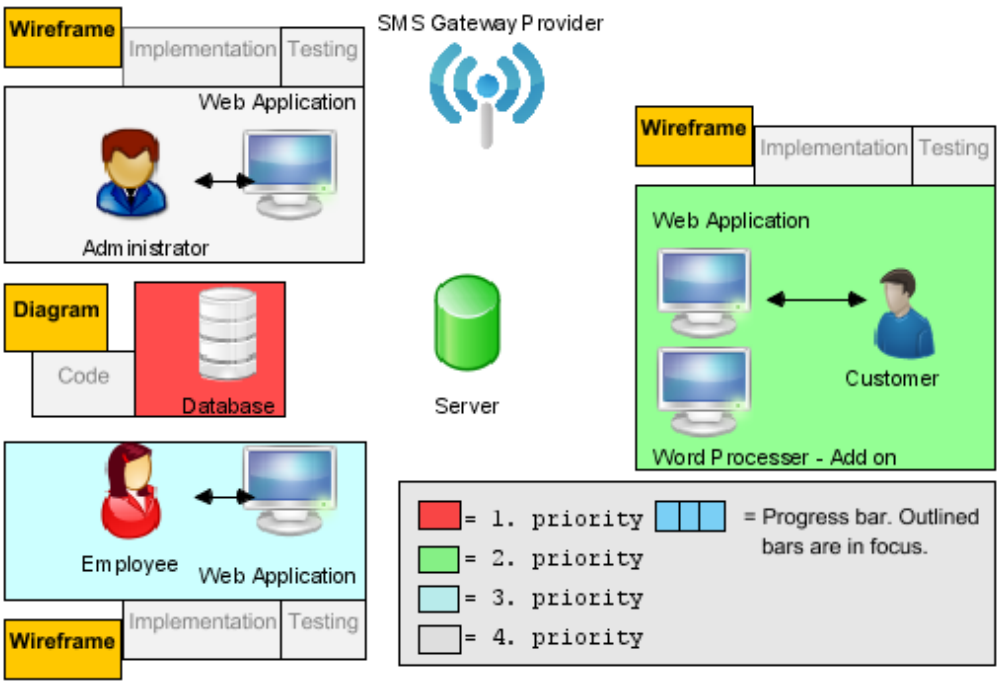


Figure 5.1: High Level System Focus Diagram

5.1.2 Sprint Backlog

Table 5.1: Backlog Sprint 1

Item	*Priority*	*Task*	*EH*	*AH*
1.1	H	Make basic wireframes for the web interface(prototype)	10	6
1.2	H	Look into macro/add-on for Word/OpenOffice	12	9
1.3	H	Make ERD for the database	10	12
TotalHours			32	27

Description for backlog table 5.1: In the first sprint we have not started implementing any functionality. The wireframes and ER-Diagram are needed before the items in the product backlog can be implemented. It could be argued that this should not be called a backlog. All the item in this backlog are high priority because they are needed before anything else can be done.

5.2 Deliverables

This section will include our final product after sprint 1, as well as some diagrams.

5.2.1 Wireframes

Because we are using Scrum, we will be developing new prototypes of the software each sprint. For this first sprint we made wireframes of the website. Wireframes are just skeletons without any functionality, which makes them very fast come up with and easy to change later on. Because of this, we had something to show the client very early in the project. Here is a screenshot of the Admin interface, more screenshots can be seen in the appendix.

Home page

Figure 5.2 shows the current home page for the system. This is the first page users see when they visit the page. Here the user can register and login among other things. The user can also change the language in which the page are presented.

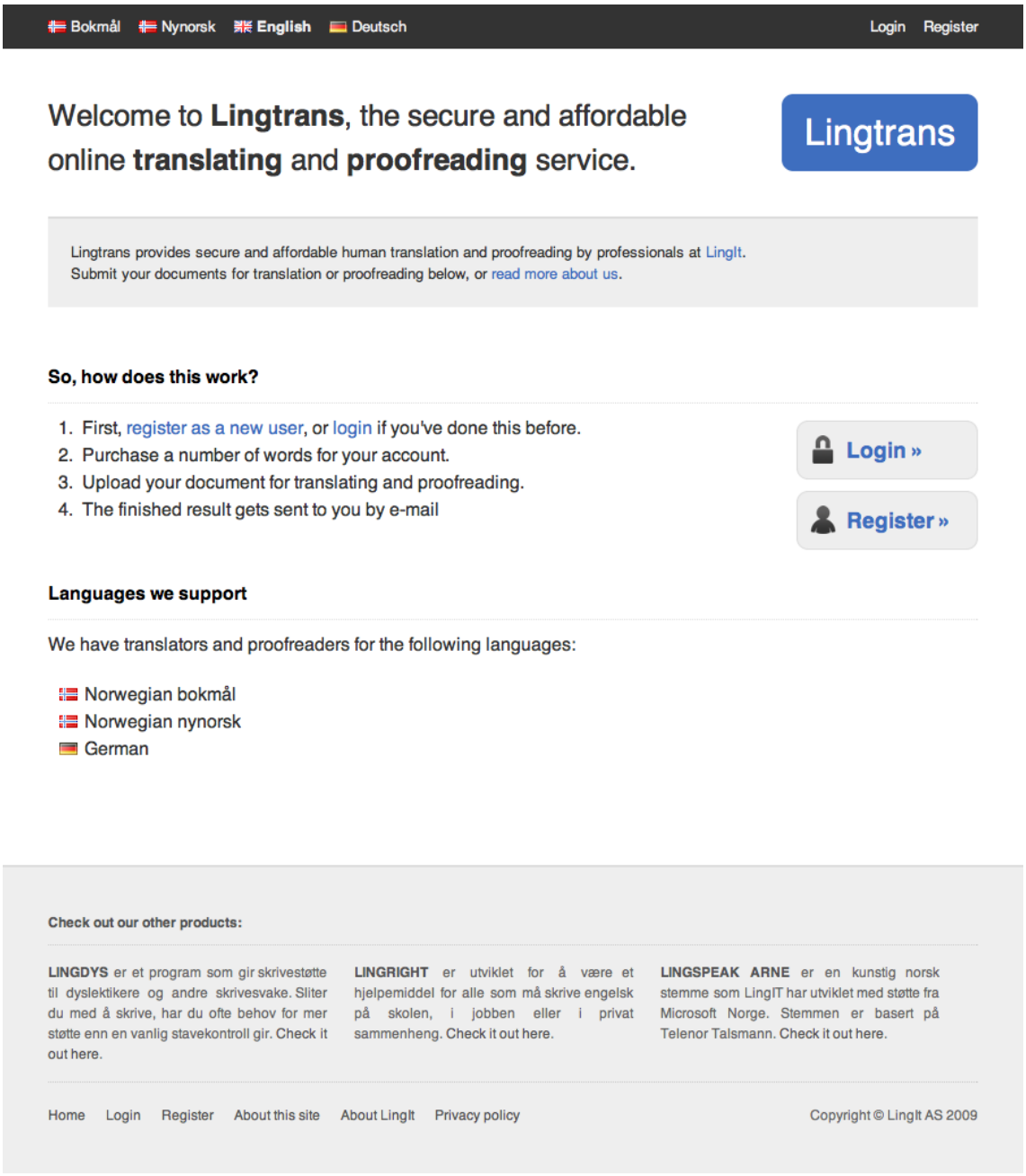

Bokmål Nynorsk English Deutsch
Login Register
Welcome to Lingtrans, the secure and affordable online translating and proofreading service.
Lingtrans
Lingtrans provides secure and affordable human translation and proofreading by professionals at LingIt.
Submit your documents for translation or proofreading below, or read more about us.
So, how does this work?
1. First, register as a new user, or login if you've done this before.
2. Purchase a number of words for your account.
3. Upload your document for translating and proofreading.
4. The finished result gets sent to you by e-mail
Login »
Register »
Languages we support
We have translators and proofreaders for the following languages:
Norwegian bokmål
Norwegian nynorsk
German
Check out our other products:
LINGDYS er et program som gir skrivestøtte til dyslektikere og andre skrivesvake. Sliter du med å skrive, har du ofte behov for mer støtte enn en vanlig stavekontroll gir. Check it out here.
LINGRIGHT er utviklet for å være et hjelpemiddel for alle som må skrive engelsk på skolen, i jobben eller i privat sammenheng. Check it out here.
LINGSPEAK ARNE er en kunstig norsk stemme som LingIT har utviklet med støtte fra Microsoft Norge. Stemmen er basert på Telenor Talsmann. Check it out here.
Home Login Register About this site About LingIt Privacy policy
Copyright © LingIt AS 2009

Figure 5.2: Wireframe Homepage

5.2.2 Add-on

In addition to the web application the client wanted an add-on for MS Word and OpenOffice so the user could proofread/translate texts without having to open a browser, log into the account and upload a file. We thought about something that would work in the following order: You select the text and click a translate button, a box will pop

up where you can enter your username and password and upload the text you have selected. We are not sure if this can be done.

MS Word

For MS Word there are macros. They are written in VBA(Visual Basic for Applications) and can be made to do a lot of things in word and we hope they can be used to upload the text for proofreading and/or translation. There is one concern about the macro: we are not sure we can protect the code from being seen by everyone. You can password protect it, but we have to look into how safe that is. Because of the security issue the client suggested to use C# for implementing the Word add-on instead of a macro. This will be more secure, but also more complicated to make. We will make the add-on as a macro to start and then we may port it to C# if the security issue is so big as we fear.

OpenOffice

This part can be written in Java. It exists some wizards to make skeleton code for this. So it should not be so hard to start with this. The group are fairly good in java. This should help us with this part.

5.2.3 ER Diagram

We use MySQL (My Structured Query Language) as the relational database management system, the program runs as a server providing multi-user access to a number of databases. MySQL is used by free software projects which require a full-featured database management system. It is also used in very high-scale World Wide Web products including Google and Facebook[21]. It is often combined with PHP, like in our case, and its popularity is closely tied to the popularity of PHP. It is written in C and C++. The SQL parser uses yacc and a home-brewed lexemizer, sql_lex.cc. It works on many different system platforms.

Libraries for accessing MySQL databases are available in all major programming languages with language-specific APIs. We are currently working using the latest version of the database, MySQL 5.1.

We are currently using 12 tables in the database, we have designed it in order to obtain the maximum efficiency in data dimension and time, all the query used in the project will be optimized and will get results really fast. The document table has 7 foreign keys to keep the consistency with the classification, customer, employee, format, deadline and language tables. We have done one table in order to know which are the abilities of the employee, translations, with also the rank of the employee in that ability. If the language_from_id and language_to_id in the document table have the same value, it means that the text has to be proofread.

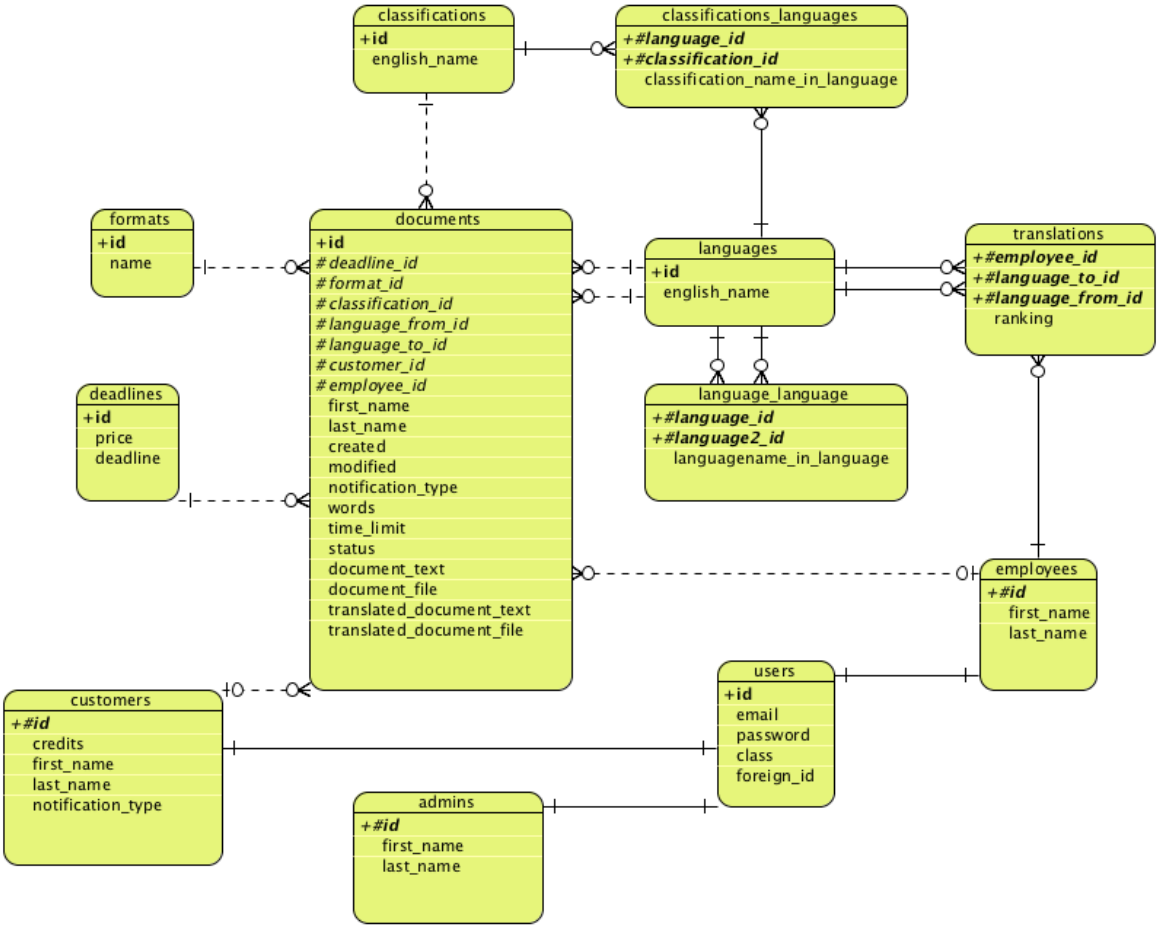


Figure 5.3: ER Diagram

The document table has also two blob fields and two text fields in order to store both the original text and file sent by the customer, document_text and document_file, and the translated/proofread by the employee, translated_document_text and translated_document_file. The Deadline table has been built because the customer can choose which kind of priority he needs.

We have built the language_language and classification_language tables in order to obtain translated text on different languages. For example the same language can be printed in more way, "italian, italiano, italiensk". The same discourse for the classification_language table, a topic can be printed in different languages, so with these tables we prevent using unwanted PHP code to manage the situation.

5.3 Sprint Evaluation

In this first sprint we planned the following implementation sprints. So far we have had three meetings with the client, and we have built up a good understanding of what it is the client wants. We have also developed the basic framework for how, when and where the upcoming client meetings will be held.

Setting up the server was the first task of this sprint. We set up source repositories for the code and the report, and installed the necessary software on each group member's laptop. We also installed Latex for editing the report, and found software suitable for UML modeling, scheduling and code editing.

In our first client meeting, they presented us their initial idea, and we had a brainstorming session regarding the product. They had not developed a detailed requirements specification, and wanted our input on how their idea should be realized. Through this brainstorming session, the client and the group arrived at a basic understanding of what the finished system would look like. Over the next two client meetings, this basic understanding was further refined and made more precise.

For the second client meeting, we created an early backlog for the project, a few descriptive UML use cases, and a few HTML wireframes, so that we could be sure the client understood how we would approach this project. The client was pleased with our progress, and remarked that the wireframes function as the first draft of the finished web site design.

For the third client meeting, we developed a preliminary ER diagram for the database. This diagram was further developed during and after the meeting, as the client had many ideas about how the database should be structured. In this meeting, we also presented our choice of PHP framework, which the client accepted. In addition, we presented our preliminary findings on how the text editor plugin that works with the site should be made.

After the third client meeting, before the start of the first implementation sprint, we finished the project backlog and the backlog for the first sprint. We have also set up the server for out live code repository, so that we can easily test the implemented code once we start the next sprint.

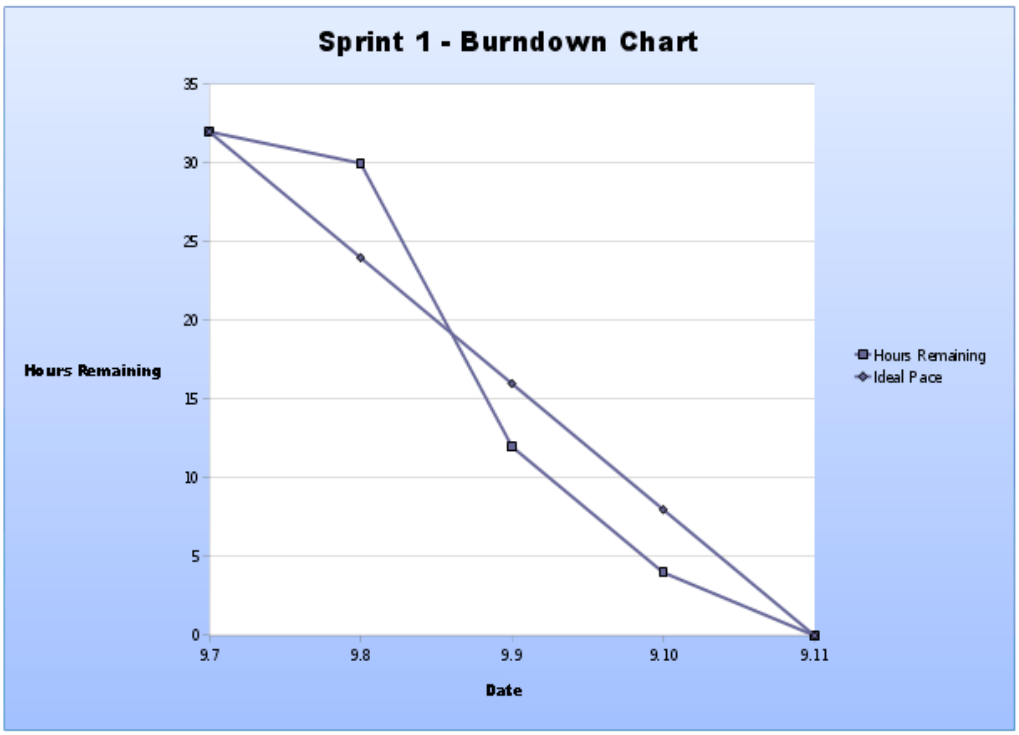


Figure 5.4: Burndown Chart Sprint One

Burndown Chart : The chart 5.4 illustrates the work left in the sprint with respect to the estimated hours. When we first started, the tasks were easier to finish than expected and thus a lot of hours were saved.

CHAPTER 6

Sprint 2

Sprint 2 is the first implementation sprint. It will last for two weeks, and will focus on high priority customer items from the product backlog. This will include the the most important features of the web application and the macro add-on for Microsoft Word.

This chapter presents the goals, backlog, delivered features and left over backlog items from sprint 1.

6.1 Sprint Planning

This section includes goals to achieve and the backlog items to be implemented in this sprint.

6.1.1 Overall Sprint Goal

The first implementation sprint will last for two weeks. Since this is our first implementation sprint, estimating the time needed for each backlog item will not be something we can do with high precision. The goals for this first sprint is to implement many of the high priority items from the project backlog. At the end of the sprint, we will have a rudimentary working prototype, with significant parts of the finished system's features. The system focus for this sprint is illustrated in the High Level System Focus Diagram 6.1 below.

If there is too little time in this sprint to finish all the sprint backlog items, they will be pushed to the next sprint, and considered when we draw up the backlog for that sprint. If there are too few items, we will select further work from the main product backlog.

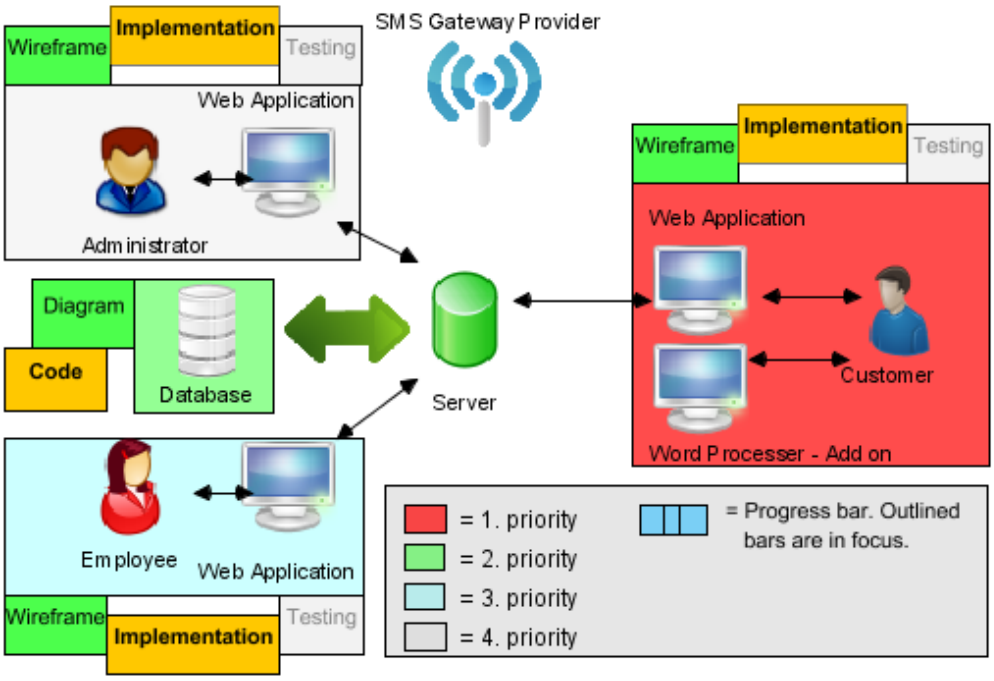


Figure 6.1: High Level System Focus Diagram

The basic data flow between the user/employee and the server can be seen in Figure 6.2. Figure 6.3 describes the server flow a bit more in detail. The blue square represents the server at Lingit. This is the way we envision how the flow of documents will eventually be. As can be seen, files are saved directly into a database and never stored as files directly on the file system. In the diagram, the database is on the same server as the web server. This is however not a requirement, the two servers can be on two different locations as long as they are able to connect to each other.

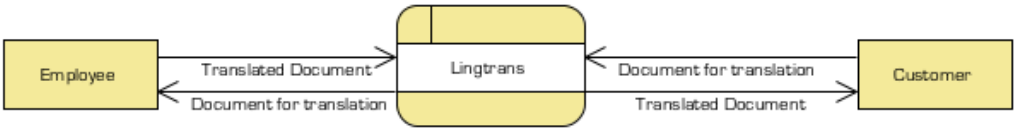


Figure 6.2: Context Level Data Flow Diagram for the Web Application

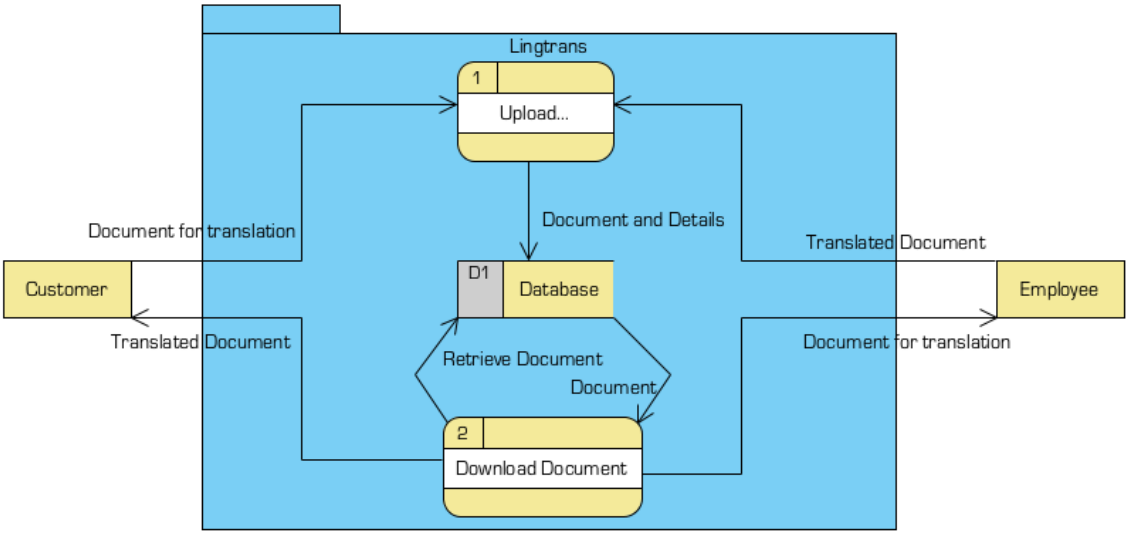


Figure 6.3: Level 1 Data Flow Diagram for the Web Application

6.1.2 Sprint Backlog

The following tables are the same as the high priority items from the product backlog, with a few items deferred to later sprints.

Table 6.1: Web Application Backlog - Sprint 2

Item	*Priority*	*Task*	*EH*	*AH*
Customers				
1.1	H	A customer should be able to register on the website with an email, password, full name and billing details	2	2
1.2	H	A customer should be able to login with the username and password after registration	1	1
1.3	H	A customer should be able to paste text into a text box and send it in for proofreading or translation	12	12
1.4	H	A customer should be able to upload Word documents and plain text files to the server for proofreading or translation	12	14
1.5	H	A customer should have the option to choose the language of the text, and the target language if this task is a translation when uploading a document	8	10

Continued on next page

Item	*Priority*	*Task*	*EH*	*AH*
1.7	H	A customer should be able to download the resulting proofread or translated documents, when logged in	12	12
Employees				
1.8	H	An employee should be able to log in if the person is already registered in the system	1	1
Administrators				
2.10	M	An administrator should be able to log in if registered in the database	2	2
TotalHours			50	54
(Items extra in original table: 1.6)				

Table 6.2: Add-on Backlog - Sprint 2

Item	*Priority*	*Task*	*EH*	*AH*
4.2	H	The customer should be able to select text in the editor and press Proofread or Translate	3	5
4.3	H	The add-in should ask the user for username and password, job deadline, text type, the source language and optionally the target language	5	5
4.4	H	The add-in should send the selected text to the server API, together with the user details	8	12
4.5	H	The web application should send an email to the user when the translation from the add-in is finished	5	5
4.6	M	The add-in should be able to check the server for results	5	5
4.7	M	The add-in should download the resulting text and show it in a window	5	5
TotalHours			21	27
(Items extra in original table: 4.6, 4.7, 4.8, 4.9, 4.10)				

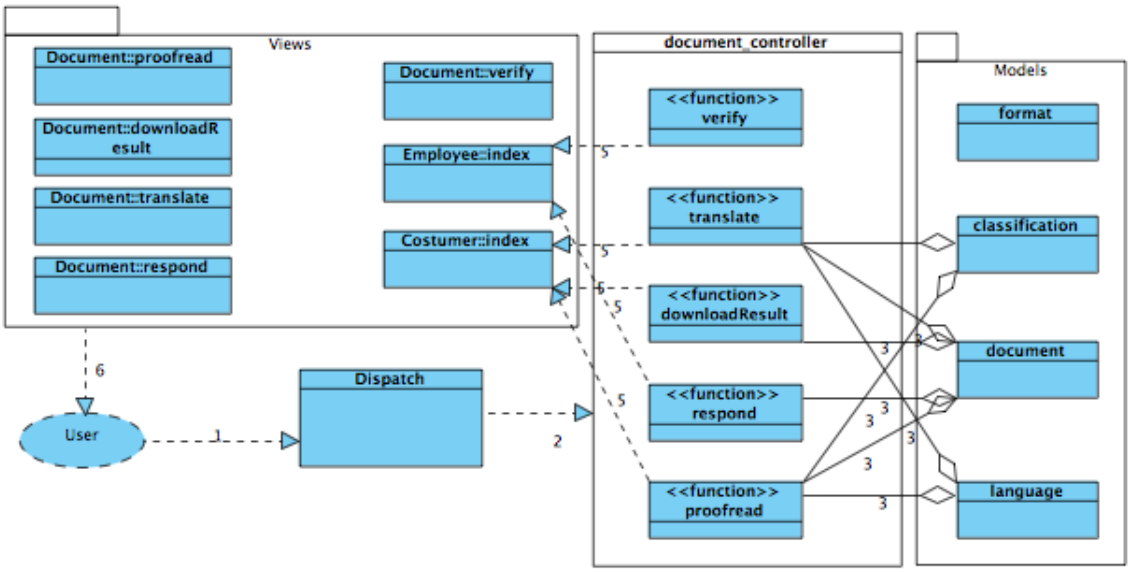


Figure 6.4: MVC Structure for the Document Controller

6.2 Deliverables

Figure 6.4 is a example of how mvc works and it is the same structure as figure 3.2. This is the document controller, which models it interacts with and which pages it sends info to. The user will be at one page, do an action and end up somewhere else. As you can see in the views, the notation Document::respond, shows that the view is in the document folder and is called respond. This is a one to one relation with the functions in the document controller. Every view page have their own function.

All the views to the left in the view box is pages that uses the functions shown in the document controller. When the document controller ask the model for information, the models will reply. This is not in figure 6.4 because it would have been too many arrows and the purpose would have been lost.

6.2.1 Class Diagram

This is the first version of our class diagram. It will most likely evolve over the next sprints.

Models

Some models correspond directly to a database table, while others do not. Those who do have the same fields as the corresponding database table. The ER-diagram 5.3 shows

all the tables in the database. The functions of each model is still not all implemented so no functions are shown in figure 6.5

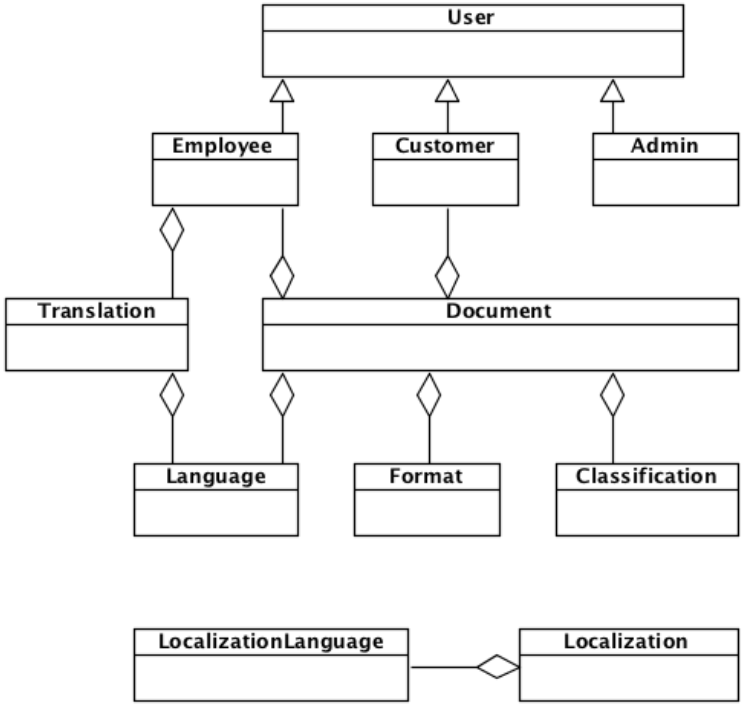


Figure 6.5: MVC Model Classes

Table 6.3: Models

Model	*Description*
User	A generalization of the common fields and methods in the Customer, Admin and Employee classes. Used for the authentication system.
Admin	Holds the admin user objects.
Customer	Holds the customer objects.
Employee	Holds Employee objects.
Translation	Describes the qualifications of each employee, i.e. which languages they can proofread in and translate from and to. Holds IDs from the Language model.
Document	The primary database table. Holds all documents, and connects documents to customers, employees and document details.
Format	Holds the different formats the site allows for uploading.
Language	Holds the different languages the uploaded documents can be written in.

Classification	Documents are classified by the customer into one of the categories held in this class, to help the employee with processing the document.
LocalizationLanguage	Languages for translating the HTML pages for localizing the site.
Lozalization	Language strings for each part of text on the site, in each of the localization languages.

Controllers

Each section of the website has its own controller.

Table 6.4: Controllers

Controller	*Description*
PagesController	Each static text page of the site, and the main index page.
UsersController	Handles users logging in and out.
CustomersController	The customer home page and other customer pages.
EmployeesController	The employee home page and other employee pages.
DocumentsController	Pages for uploading and responding to documents.
AdminsController	Administrator pages for manipulating the database and seeing site activity.
LocalizationsController	Controller for administering the localization languages and localization strings.
ApiController	The access point for the text edit plugins, does not render any views.
AppController	Common setup methods for all controllers.

Libraries

So far, the system only uses one external library:

Table 6.5: Libraries

Library	*Description*
XmlRpc	A library used for communicating with the text-editor plugins through the XML-RPC protocol.

6.2.2 Web Application Deliverables

As seen in the sprint backlog for the web application, this sprint focused on implementing the high priority features for the web application. These features lays the foundation for work in the upcoming sprints. See figure 6.4 and 6.5 for a better understanding of how the classes and functions are connected.

Item 1.1, Item 1.2, Item 1.8, Item 2.10: The user authentication system was the first implemented part of the system. The system is based on four models (and database tables): `User`, `Customer`, `Employee` and `Admin`.

`User` is a polymorphic model which contains the login details for users of all classes. The other models contain specific data for the different classes of users. This means we can extract all common login and registering functionality to one model, and use the same login page for all users.

To authenticate and keep users logged in, we use CakePHP's built in `Auth` component, which allows us to specify which pages should be accessible when logged in and out. On top of this, our system takes care of authenticating users by their class, denying access where needed.

Each controller has the following permission specification in their `beforeFilter`, which is called before any page in rendered:

```
# permissions
$this->allow(array(
  'Logged_out' => array('register'),
  'Customer' => array('index', 'buy', 'edit', 'sendEmail'),
  'Employee' => array(),
  'Admin' => array('edit', 'delete')
));
```

Each class of users get access to different methods, and they all get access to the ones specified in the `Logged_out` array.

Item 1.3, Item 1.4 and Item 1.5: The document upload feature may be the most important feature of the whole system. Almost all other features is made with some connection to this.

When a customer wants to upload a text for proofreading or translation he/she has two choices. Either upload a document file or paste a text in a text box on the page. When uploading some details about the document should be set for easier assignment to the employee. In addition some time limit has to be set. When the user press the upload button the text is saved in the database. Documents are saved as blobs and texts in the text box as texts in the database. It may be necessary to make the saving of texts

a bit different in the future because it is slow to save whole texts in the database, but for now that is fine.

The logic behind the upload connects the document to the right customer table, employee table and language tables to mention the most important ones.

The proofread sequence diagram(figure 6.6) is the exact same as the sequence diagram for the uploading of translate except that you start in the translate view instead of the proofread view. See figure C.1 in the appendix. Therefore, just proofreading will be shown here. Figure 6.6 is the sequence diagram for the use case shown in figure 4.3. The proofread view contains a form where the customer uploads a document or inputs a text and set some options about that text. The customer will then have to submit the form and the controller takes care of finding a suiting employee for that task. Next the document is saved in the database or the saving will fail for some reason. The customer will get a message in both outcomes.

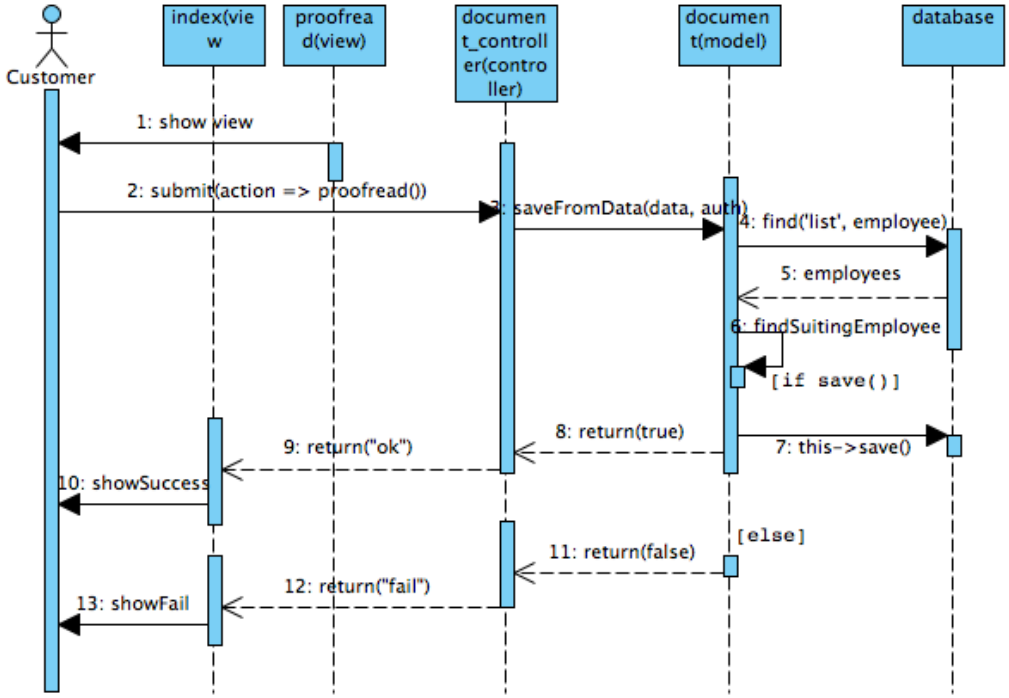


Figure 6.6: Upload for Proofread Sequence Diagram

Explanation for figure 6.6:

1. The proofread view is shown to the customer(the customer must have pressed the proofread link before. The data flow does not just start here.)
2. The customer inputs the informations he or she wants in the form and submits the form.

3. The customer controller lets the customer model do the logic on the data.
4. The customer model finds a list of all employees.
5. The database returns a list of all employees.
6. The model finds the most suiting employee and saves the it with the data.
7. If the data can be saved it is saved in the database.
8. True is returned to the controller.
9. Create view with a message indicating that the upload was a success.
10. Show the success to the customer.
11. If the data can not be saved false is returned to the controller.
12. Create a view with a message indicating that the upload was a failure.
13. Show the failure to the customer.

Item 1.7 When the customer has uploaded a document the employee has to be able to download it and the customer has to be able to download the finished result.

When the customer downloads the result it is transformed from a blob to the filetype it was uploaded in and automatically downloaded to the machine. If the text was uploaded in the text box, a txt-file with the result will be downloaded. Documents can be downloaded as many times as wanted.

Figure 6.7 is the sequence diagram for downloading the finished documents for the customer. For more information see the use case 4.3. In the index view all documents for that customer are being shown. The customer can click on any of them and download them to their personal computer.

Explaination for figure 6.7:

1. The customer press download on one of the finished documents.
2. The document controller finds the given document by using the id.
3. The document is returned.
4. The document controller ask the document model to return the result.
5. The model return the blob if it was uploaded as a file or the text if it was uploaded as a text.
6. The document controller constructs the file from the data given by the model.
7. The file is automatically downloaded to the customers computer(The most browsers will ask the user for a confirmation of the download)
8. The index view is created.

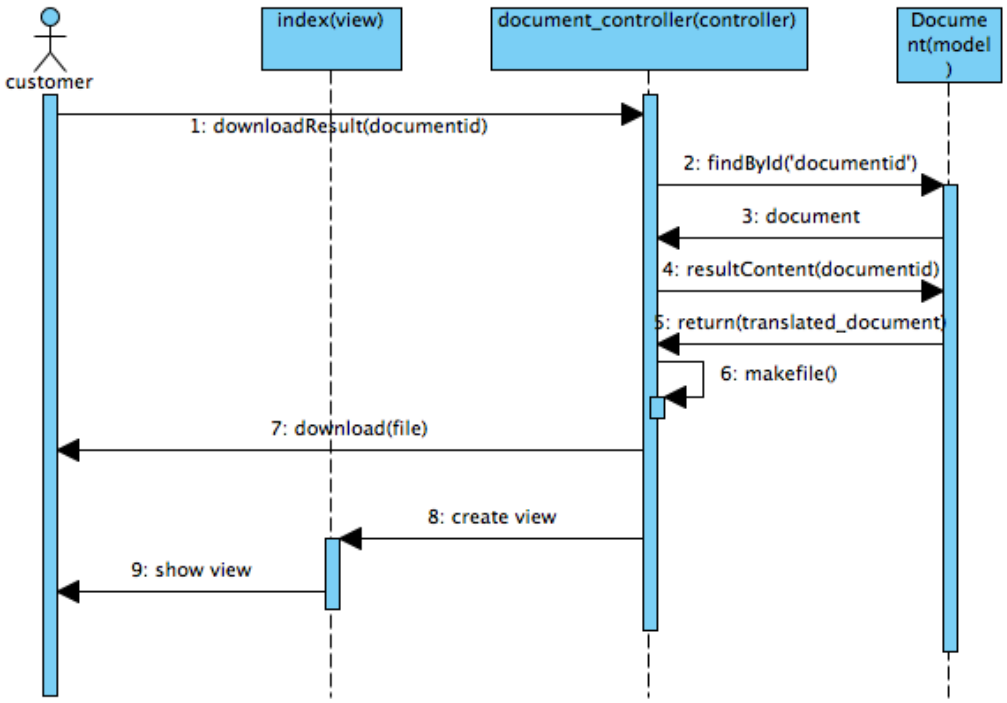


Figure 6.7: Download Finished Document Sequence Diagram

9. The index is shown to the customer.

6.2.3 Microsoft Word Macro for Windows

Implemented Features

In the figure 6.8 is shown the use case diagram corresponding to the use cases of the Word Macro:

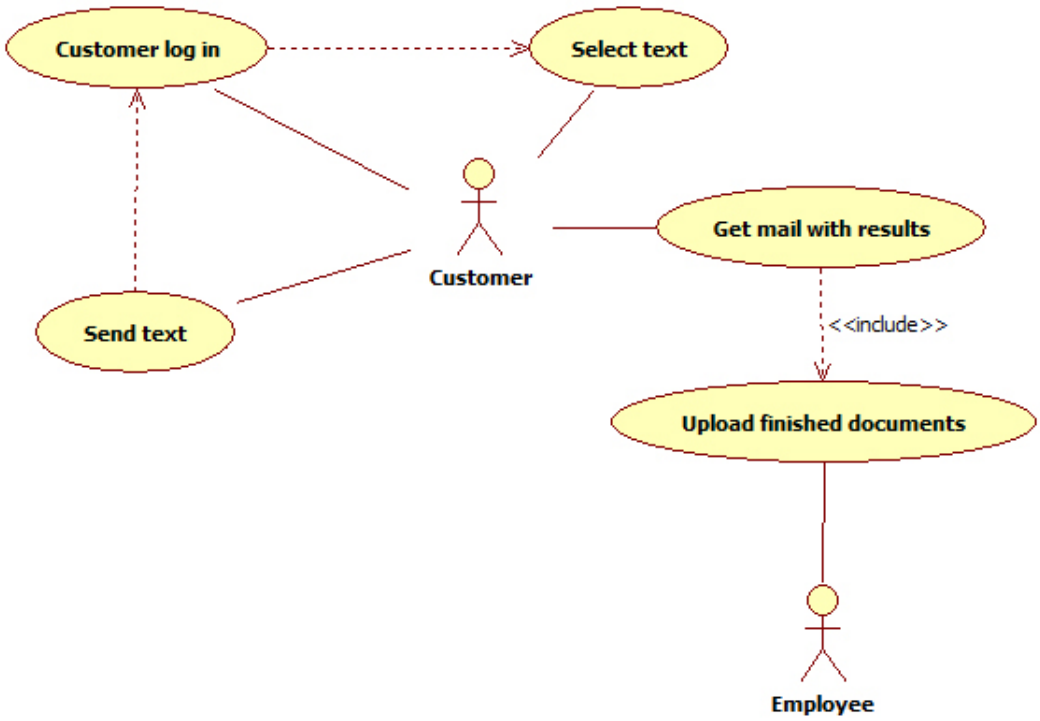


Figure 6.8: Use case for the Microsoft Word Macro

There are basically three use cases linked to the use of the macro: Select Text (table 6.6 and figure 6.9), Customer Log In (table 6.7 and figure 6.10) and Send Text (table 6.8 and figure 6.11). The other use cases in relation with the employee (directly or indirectly) are here to show the interaction between the macro and the server-side system, which is the core of the system.

Table 6.6: Select Text

Name: Select text	**Objectives**: Allow the user to select the part of text he/she wants to sent to his/her account.
Primary actors: A customer **Secondary actors**: None	**Dates**: 31.09.09 **Responsible**: Brian Mathé **Version**: Sprint 2
Preconditions: Microsoft Word application is launched and the user has clicked one of the buttons.	**Postconditions**: The form is displayed to the customer

Scenarii: - The user has selected the text he/she wants to send to the service: the form to enter the user informations is displayed and the user is allowed to put his account/password to connect to the service.
- The user hasn't selected any text: a message box is displayed asking him to select the text he/she wants to send to the service first.

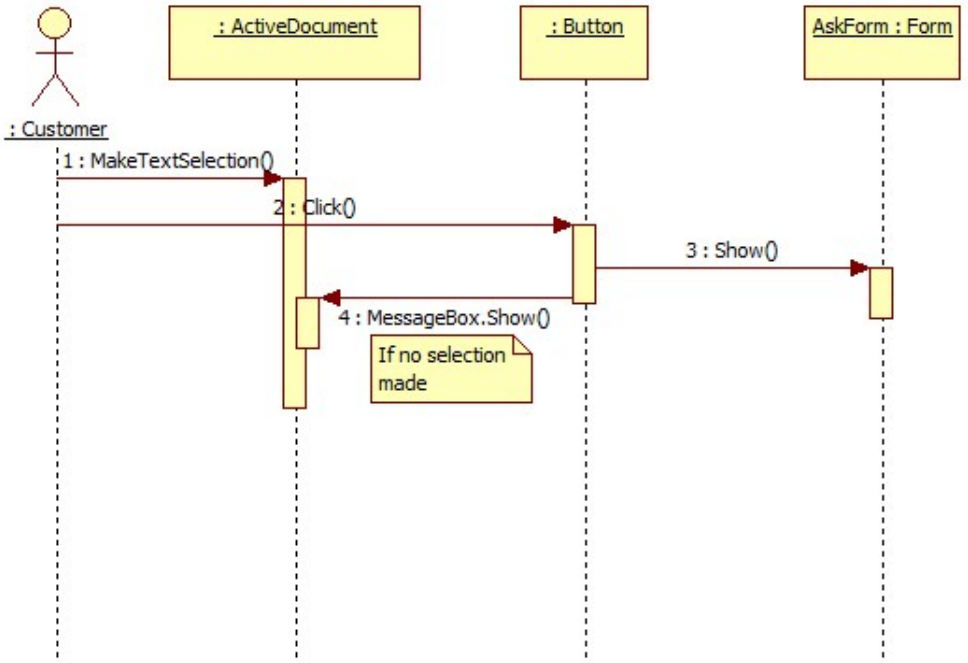


Figure 6.9: Select Text Sequence Diagram

Explanation for figure 6.9:

1. In the active word document the customer selects the text he/she wants uploaded for translation or proofreading.
2. The customer presses the proofread or translation button in word to start the process of sending the text to the server.
3. Show a form where the user most insert information about the document and user details.
4. If there is no text selection a message box will alert the customer.

Table 6.7: Create Account Description

Name: Customer Log In	**Objectives**: Identify the user sending text from the macro and allow him to send texts.
Primary actors: A customer **Secondary actors**: None	**Dates**: 31.09.09 **Responsible**: Brian Mathé **Version**: Sprint 2
Preconditions: The user has filled in the form with his/her correct personal e-mail and password	**Postconditions**: The customer can enter the informations about his/her text in the dedicated form.
Scenarii: The user informations are put into an XML string which is sent to the server with a specific method from the API for logging him in. The server then sends back an XML string containing this user's news. The add-in then extracts the useful information from this string to display it in a MessageBox. Afterwards only the user has to specify in combo boxes the informations about his/her document: input language, output language (in the case of a translation) and the deadline.	

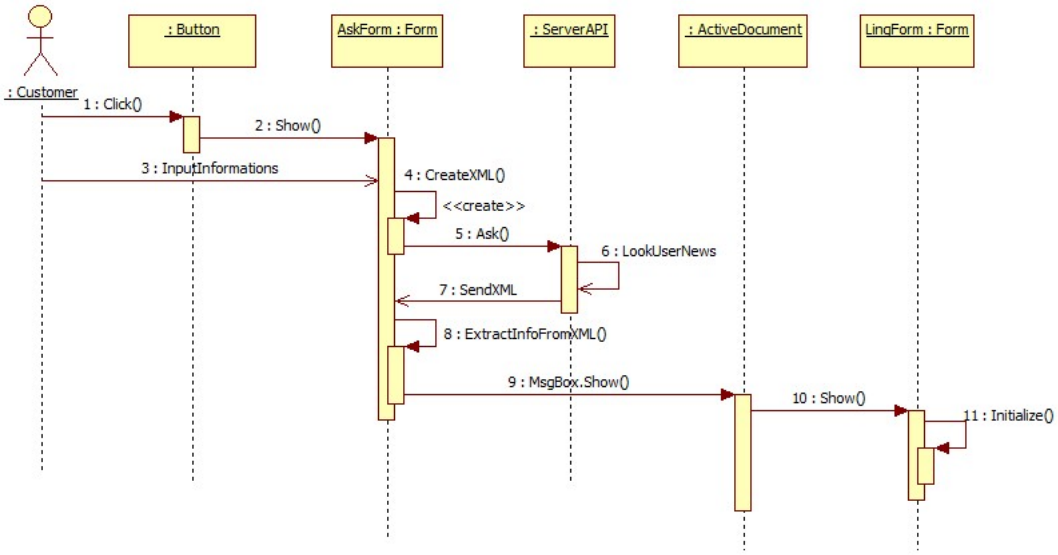


Figure 6.10: Customer Login Sequence Diagram

Explanation for figure 6.10:

1. The customer press the buttom for proofread or translation.
2. Show a form where the user most insert information about the document and user details.
3. The customer inputs the information needed. Login information and the information about the document.
4. A xml-string is made.
5. The askForm sends the information to the server api.
6. The server api checks the user information.
7. The server api send xml back to the macro.
8. The askForm extracts the information from the xml.
9. The askForm create a message from the information given from the server and sends it to the active document.
10. The active document send the message to the LingForm which shows the message to the user.

Table 6.8: Send Text

Name: Send text	**Objectives**: Identify the user sending text from the macro and allow him to send texts.
Primary actors: Customer **Secondary actors**: Employee	**Dates**: 31.09.09 **Responsible**: Brian Mathé **Version**: Sprint 2
Preconditions: The user has filled in the informations about his/her document	**Postconditions**: A new job has been created in the database.
Scenarii: The use press the OK button of the document informations form. All these informations are then put in a XML string which is sent to the server in a dedicated method.	

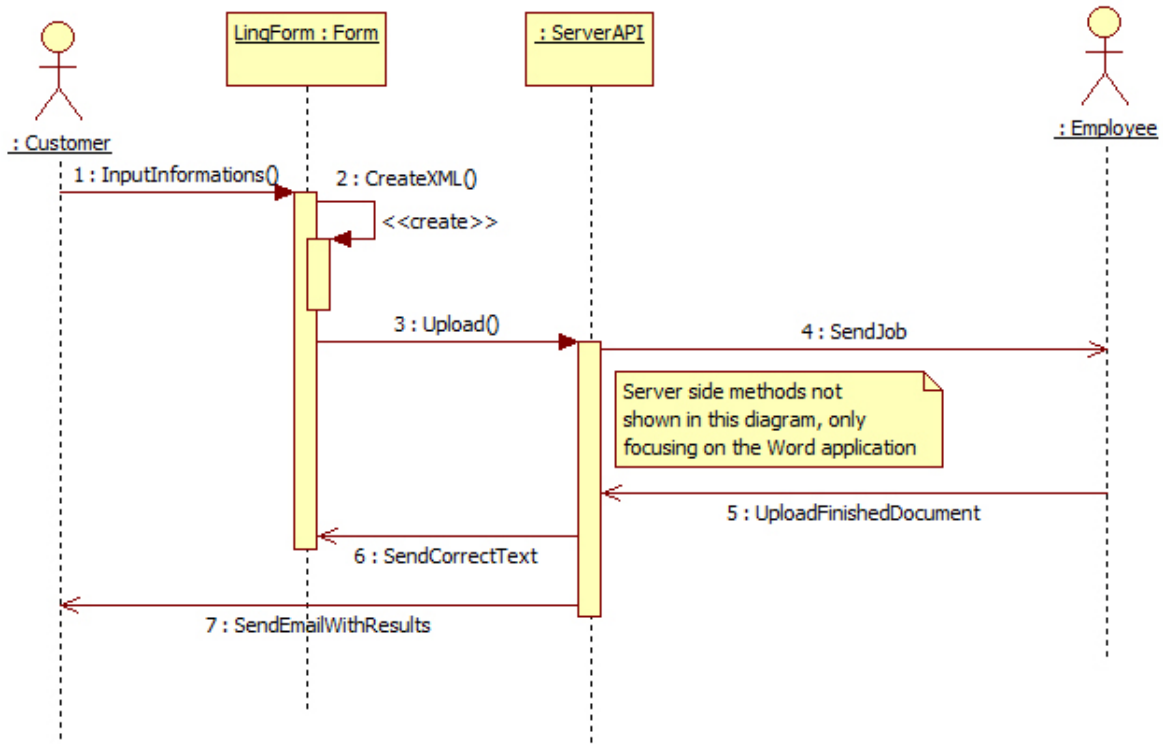


Figure 6.11: Send Text Sequence Diagram

Explanation for figure 6.11:

1. The customer inserts information in the LingForm about the document.
2. A XML-string is made from the selected text.
3. The xml-string is sent to the server API.
4. The server assigns a employee to the task.
5. After some time the employee uploads the finished document.
6. The next time the user logs onto word the add-on will search for finished documents.
7. The finished document is also sent to the users e-mail address.

How To Use the Service

The utilization of this service is quite easy. The user select the portion of text he/she wants to be treated by Lingit. He then loads a macro named Lingtrans, which activates a form in which the user must specify: e-mail, password, the job type (translation

or proofreading), the original document language, the deadline and in the case of a translation, the output language wished. Then, by clicking the OK button, all the information is sent to the server and automatically treated by the API on the server.

We assume in this version of the system that the user will receive the translation/proofreading by e-mail before the deadline he/she specified in the form and receive a notification by SMS if wished when registering.

6.3 Sprint Prototype

This section presents the finished prototype for this sprint. It is what we show the client at the review meeting. The section is separated in two parts, Web application and Word Macro.

Web application: In this sprint we focused on the customer user and the registering and logging into the page. The login was implemented for all users. Each with different start pages. The customer user could also upload documents and text in this sprint. They can also chose the language they are uploading the document in and what they wanted it translated to. All this are saved in the database. These are the most important features implemented this sprint.

Word Macro: In the end of this sprint we had a working version of the macro which could send the document to the server. This means that the user can open word, write some text and then choose, from word, to send it to the server. This will also request the user for a username/password.

6.4 Client Feedback

In the end of the sprint we had a small presentation of what we had done so far. The section is split into the web application, the code, the add-on and other. **The web application**

- The customer should not be able to submit documents without buying words first. This will only be the first time the customer uses the service. This is because LingIT wants to know that the customer is serious.
- The you submit some information in a form that is incorrect, it should be stated what is wrong instead of just telling the customer that something is wrong. This makes it more user friendly.
- It should be possible to specify the title of the text when you paste in text in the text box when you upload. At the moment the application just takes the first

word of the text pasted.

- The client was overall very pleased with what we had done with the web application.

The code

- We should take a look at how CakePHP behaves when it comes to encapsulation. The code should be strictly object oriented, and should therefore follow the standards of encapsulation.

The add-on

- What we have of the add-on so far looks good, but we should try making it a bit more user friendly.
- It was agreed that the add-on for OpenOffice should be done later. This is because we want to ensure that we have at least one working addon in the end.

Others

- The client wanted the reworked ER-diagram as fast as possible to ensure a correct and valid structure.
- LingIT wants a administrator username and password so they can log in to the system and see what we are doing.

6.5 Sprint Evaluation

This was the first sprint where we actually started implementing the features from the project backlog. We started the sprint by setting up a sprint backlog, which contains tasks to be implemented during the two weeks a sprint lasts. Because none of the team members had any previous experience with Scrum, this was a difficult task and we were never really sure about how many items that should be included. We ended up with most of the high priority items, which later on proved to be a bit optimistic. The learning curve for the PHP framework of our choice also proved to be a bit steeper than first anticipated, and demanded more time than expected. The first week was therefore basically dedicated to learning CakePHP and PHP itself.

There was a lot of important tasks completed in this sprint. First of all, we got the framework up and running and suited for our needs. This was a crucial step before the actual implementation, and therefore got a lot of attention. Second of all, a lot of the basic functionality like the user management system with registration, authentication and user classes was implemented. This will be needed in later sprints in order to correctly test functionality and assure that everything is working the way it should. These were both big tasks which proved to take a lot of time to finish.

Because of the lack of experience with Scrum in the project group, the planning process was a difficult task, and none of us really felt that we knew what we were doing. We ended up with a sprint backlog containing more items than we could manage to complete, and did not really consider the fact that we could face problems.

One of the problems that occurred during the implementation phase of this sprint was that some of the functionality we were to implement depended on other functions. This was a problem because some tasks ended up being only partly implemented, and sometimes group members had to wait for other group members to finish. This is a product of poor planning and could easily be avoided in future sprints by evaluating the functionality a bit more thorough.

Even though this sprint did not go as well as we had planned, and we faced some problems, we ended up with the functionality implemented. This was mostly due to hardheaded and optimistic group members. This first sprint gave all of the group members, who had no previous experience with Scrum, a lot of insight into the process. We especially feel that we have learned a lot about planning, which was basically unknown to us before this sprint.

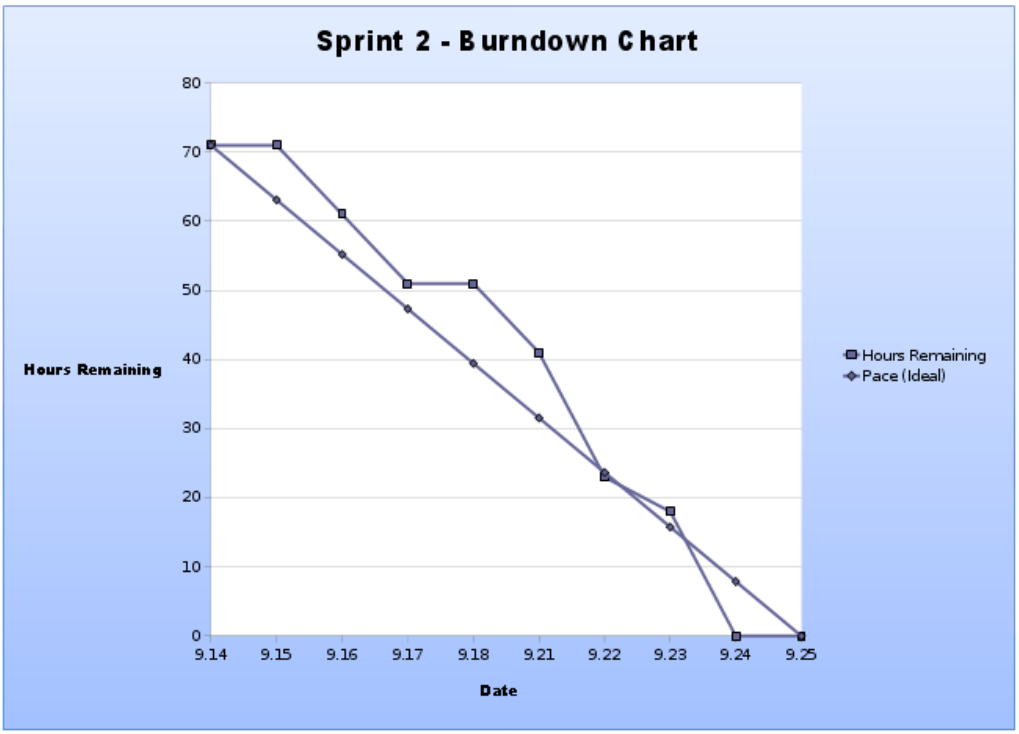


Figure 6.12: Burndown Chart Sprint 2

Burndown Chart : The burndown chart 6.12 illustrates the work left in the sprint with respect to the estimated hours.

CHAPTER 7

Sprint 3

Sprint 3 is the second implementation sprint. It will last for two weeks, and will focus on medium priority items from the product backlog, as well as left over items from the first implementation sprint.

This chapter presents the goals, backlog and delivered features.

7.1 Sprint Planning

This section will explain the overall goals we have for our this sprint, and the backlog items we will implement.

7.1.1 Overall Sprint Goal

After the first implementation sprint, we have a solid foundation for building the rest of the features. Since most of our high priority items are now implemented, we move on to the medium priority items in the product backlog. The system focus for this sprint is illustrated in the High Level System Focus Diagram 7.1 below.

As seen in the sprint backlog, there has been set lofty goals for this sprint - there is an extensive backlog to go through. Any items still left after this sprint will be pushed to the next sprint's backlog.

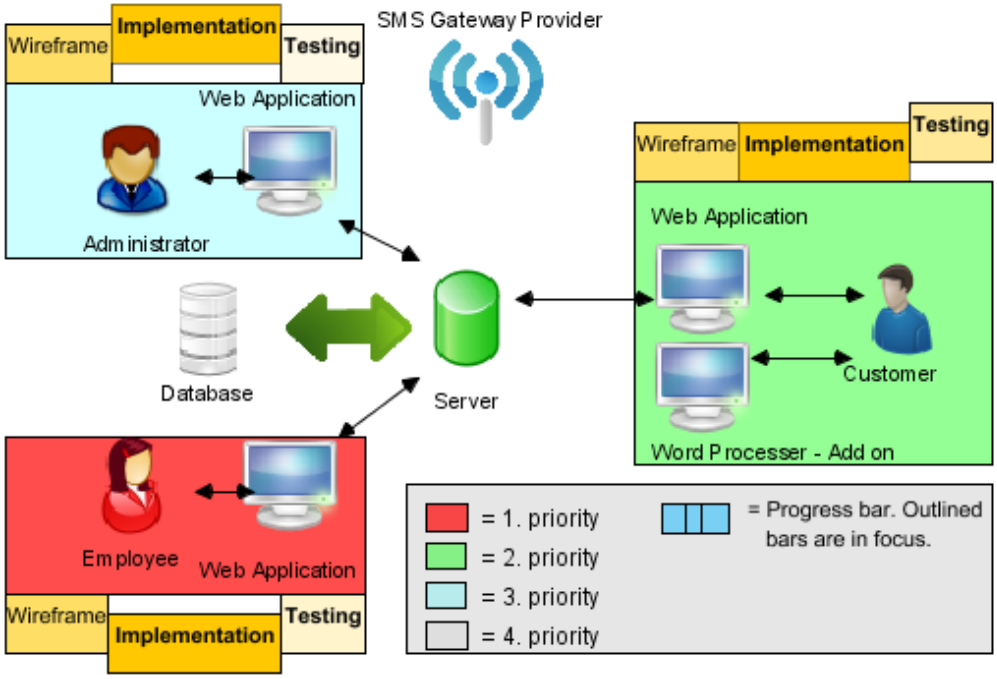


Figure 7.1: High Level System Focus Diagram

7.1.2 Product Backlog Changes

The macro add-on for Microsoft Office was completed in sprint 2 (See backlog items in table 6.2). At the sprint 2 review meeting with the client they stated that they were interested in a version of the add-in written in the programming language C#. This resulted in additional product backlog items 7.1 and non-functional requirements 7.2.

Table 7.1: Changes to Product Backlog - Added Items

Item	*Priority*	*Task*
4.9	L	The MS Word macro add-on should be ported to C#
4.10	L	The add-on should be ported to work in the latest version of OpenOffice Writer

The complete Product Backlog is found in table 4.4.

Table 7.2: Changes to Non-functional Requirements

Item	*Task*
4.1	It should be able to install the add-on with an executable.
4.2	The code should be protected from everyone to see

The complete description of the non-functional requirements is found in table 4.5.

7.1.3 Sprint Backlog

In this sprint, we focus on the medium priority items from the product backlog. We also have a few high priority items left over from the previous sprint.

Table 7.3: Web Application Backlog - Sprint 3

Item	*Priority*	*Task*	*EH*	*AH*
Customers				
1.6	H	A customer should be able to buy more word credits, which are needed to get a text processed, when logged in	2	2
2.1	M	A customer should be able to upload OpenOffice and PDF documents for translation and proofreading	1	3
2.2	M	A customer should have the option to select a classification of the uploaded text, to further guide the employee in processing the document	3	3
2.3	M	A customer should be able to select if he or she wants finished documents emailed back to their address.	3	3
2.4	M	A customer should be able to see the status of any document he or she have previously submitted, when logged in	1	2
2.5	M	A customer should be able to choose the deadline for an uploaded text	3	3
2.6	M	Each page the customer sees should be available in languages the web application supports	4	4
Employees				
1.9	H	An employee should be able to see assigned jobs and possible jobs he or she can take, after logging in	2	6
1.10	H	An employee should be able to download texts assigned to him or her	3	7
1.11	H	An employee should be able to upload finished results and connect these to the original document	2	6

Continued on next page

Item	*Priority*	*Task*	*EH*	*AH*
2.7	M	An employee should be able to set his or her upcoming availability when logged in	2	2
2.8	M	An employee should be able to set his or her skill sets in different languages, when logged in	2	2
2.9	M	An employee should be assigned texts automatically by the system, based on the employees availability and skill set	4	4
Administrators				
2.11	M	An administrator should be able to see, edit and delete customers, employees and other administrators, when logged in	4	4
2.12	M	An administrator should have the possibility to send emails to employees and customers, when logged in	1	1
2.13	M	An administrator should be able to see the status for all pending jobs, when logged in	1	1
2.14	M	An administrator should be able to see all the previous documents and finished texts, when logged in	2	3
TotalHours			41	56

Table 7.4: Add-on Backlog - Sprint 3

Number	*Priority*	*Task*	*EH*	*AH*
4.9	L	The MS Word macro add-on should be ported to C#	-	-
4.10	L	The add-on should be ported to work in the latest version of OpenOffice Writer	-	-

7.2 Deliverables

This section will include the functionality implemented during this sprint.

7.2.1 Web Application Deliverables

While the last sprint only focused on functionality for users and employees, this sprint focused more on the overall functionality of the system. In this section we will show in detail the most important features.

Administrator functions: Item 2.11, 2.12, 2.13 and 2.14: When logged in, an administrator should be able to view or edit all customers details. Figure 7.2 shows the sequence diagram for this task. The use case is found in figure 4.3. The admin sees all customers in a table, with limited information like for example name, email and phone number. Clicking the name queries the Customer-controller for that specific customer, which in turn queries the model. The controller replies to the administrator with a view for editing or viewing all registered information on this customer. The communications with the database are not shown in order to keep the readability of the diagram, and it is not necessary to draw them since they are always the same built-in functions of the framework in every diagram.

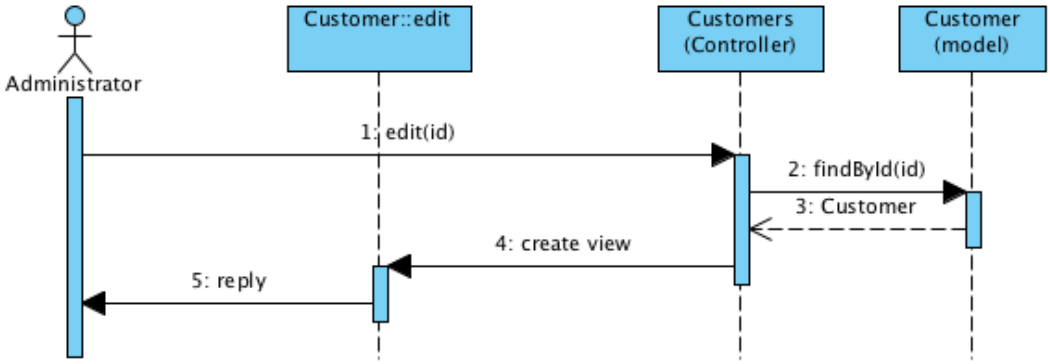


Figure 7.2: Viewing or Editing a Customers Details

Explanation for figure 7.2:

1. The administrator clicks the edit button and sends an http request to `customers/edit/id`.
2. The function `edit()` is called on the Customers-controller.
3. The controller calls `find()` on the model to get information on the requested customer.
4. A query is run on the database, which returns information on the customer.
5. Information is sent back to the controller.
6. The controller creates a view and responds to the request sent by the administrator.

When logged in, an administrator should be able to delete a specific customer. Figure 7.3 show the sequence diagram for doing this task. The use case is found in figure 4.3.

This task is done directly from the table of customers. When the administrators clicks the button for deleting a customer, the request is sent to the customers controller, which in turn finds the customer through the customer model and deletes him from the database. The administrator is then returned to the administrator index with a confirmation message.

Explanation for figure 7.3:

1. The administrator clicks the delete button and sends an http request to `customers/delete/id`.
2. The function `delete()` is called on the Customers-controller.
3. The controller calls `delete()` on the model to get information on the requested customer.
4. A query is run on the database, which removes the customer.
5. The model returns true to the controller.
6. The controller creates a view and responds to the request sent by the administrator.

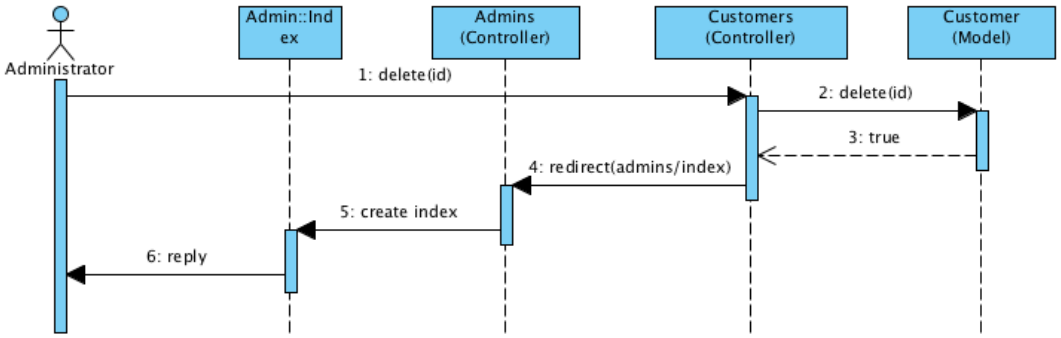


Figure 7.3: Deleting a Customer

When logging in, administrators are presented with tables for customers, employees and administrators. This means that the administrators index view queries the models Customer, Employee and Administrator to get information from the database. From these tables the administrator can edit information or delete users (except himself). The links for deleting and editing users are formed like this: `controller/delete/id` and `controller/edit/id`, respectively. This makes a call to the functions `delete()` and `edit()` in the corresponding controllers. Each customer and employee are presented with their email-address, allowing administrators to send mail to them by clicking on the email. (Use case for this if found in figure 4.30)

The administrator document view queries the Document model and is able to show the history of completed documents and those in progress. The communication with the database is not shown in order to keep the readability of the diagram, and it is not

necessary to draw them as they always are the same built-in functions of the framework in every diagram.

Item 1.6: When logged in, the customer should be able to buy more words. Figure 7.4 shows how this works in detail. This is the sequence diagram for the use case in figure 4.3. It starts with the customer looking at the index page or some other page with a link to the buy view. At the buy view the customer can select the amount of word he or she wants to buy and submits the form. The controller then updates the word amount for that customer. The communication between the model and the database is left out to keep it simple.

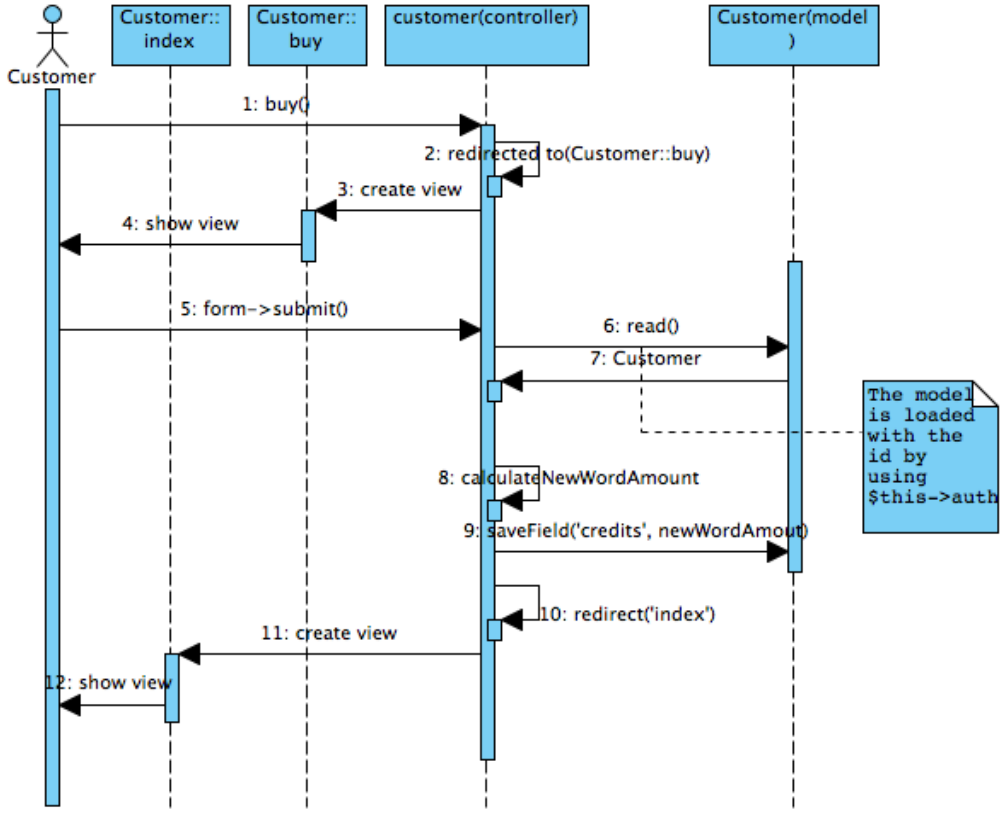


Figure 7.4: Buy More Words

Explanation for figure 7.4:

1. The customer clicks a buy link on the index page and the message is sent to the customer controller.
2. The customer controller redirects the customer to the buy view.
3. The buy view is created.

4. The buy view is shown to the customer.
5. The customer selects the amount he/she wants to buy and sends the form to the controller.
6. The customer controller gets the customer from the model by using the id.
7. The model sends the right customer back.
8. The new word amount is calculated.
9. The customer with new information is saved.
10. The customer controller redirects to the index view.
11. The controller creates the index view.
12. The view is shown to the customer.

Item 2.4: When the customer is logged in all documents are shown. They are separated in two tables; not finished and finished. Figure 7.5 shows this. The use case is found in figures 4.3 and 4.3. When the index view is loaded the controller asks the document model for finished and unfinished documents for that user and show em in the view.

To keep it a bit simple the communication between the models and the database are left out of this diagram. Explanation for figure 7.5:

1. The customer with id gets to the index page.
2. The controller ask the document model for the documents that are finished.
3. The document model returns the documents.
4. The controller ask the document model for the documents that are not finished.
5. The document model returns the documents.
6. The not finished documents are saved in a variable that the view can use.
7. The finished documents are saved in a variable that the view can use.
8. The view shows the documents.

Item 2.6: The webpage should be available in different languages. This is done by localization. Each page has flags shown at the top of the page. The user can then press one of these to change language. Figure 7.6 shows how it works when you set a language. The communication with the database is not shown in order to keep the readability of the diagram, and it is not necessary to draw them as they are always the same built-in functions of the framework in every diagram.

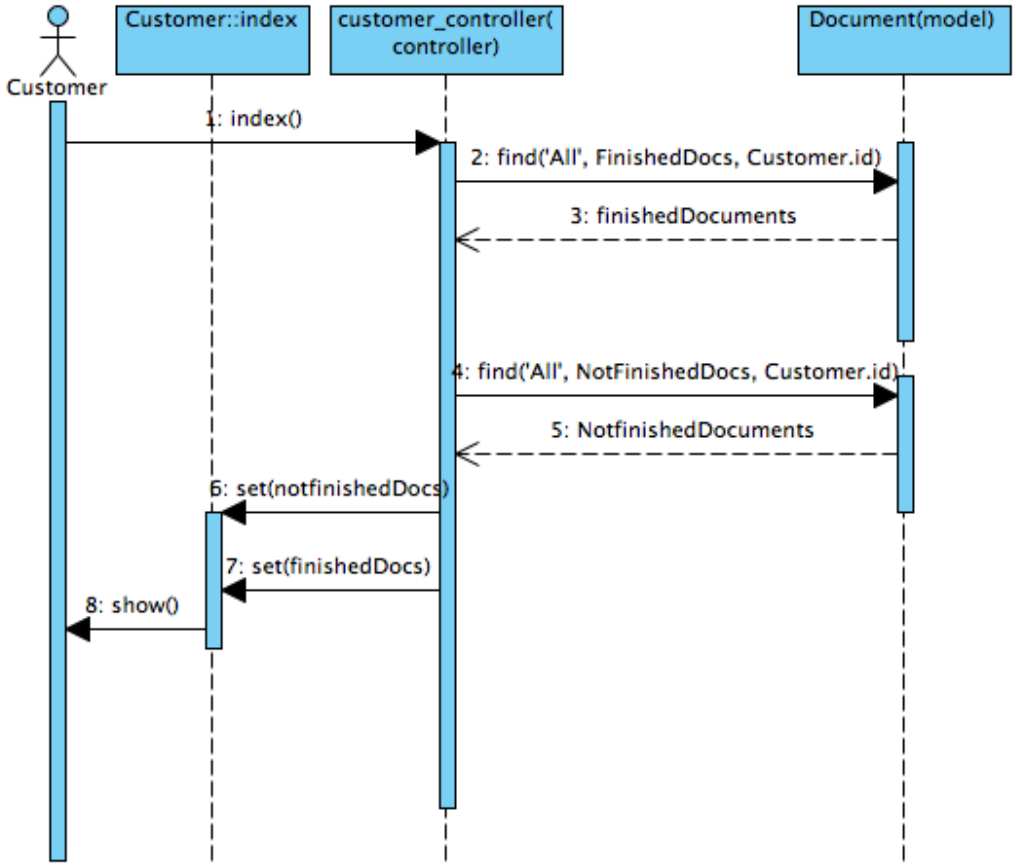


Figure 7.5: Show Finished and Not Finished Documents

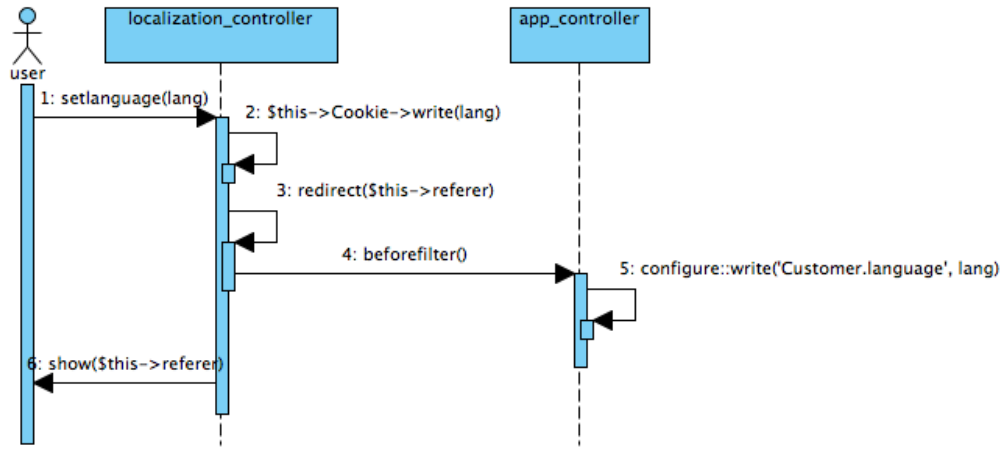


Figure 7.6: Set Language

Explanation for figure 7.6:

1. The customer presses one of the available languages at one page.
2. The cookie is updated with the given language.
3. The localization controller redirects from the page it was on when the customer wanted to change the language.
4. Every controller extends the app controller so the before filter is called.
5. Configure is a global configuration in which you can set values. The language is set in it.
6. The page the customer is on is shown with the new language. The view reads the configure to find out which language to use.

Communications with the model are not explained in the diagram since they are built-in functions of the framework.

Employee functions: Item 1.9, 1.10 and 1.11:

The employee page is based on four models: `Employee`, `Document`, `Language` and `Translation`.

When an employee log in to the system the employee page is showed up in which he can see his translating and proofreading abilities.

The employee can also see the documents that he is currently working on, with the relative deadline, and the documents that fit his abilities and that are not assigned to anyone, in order to have the possibility to add them to his tasks.

The Employee can remove a task from his tasks list.

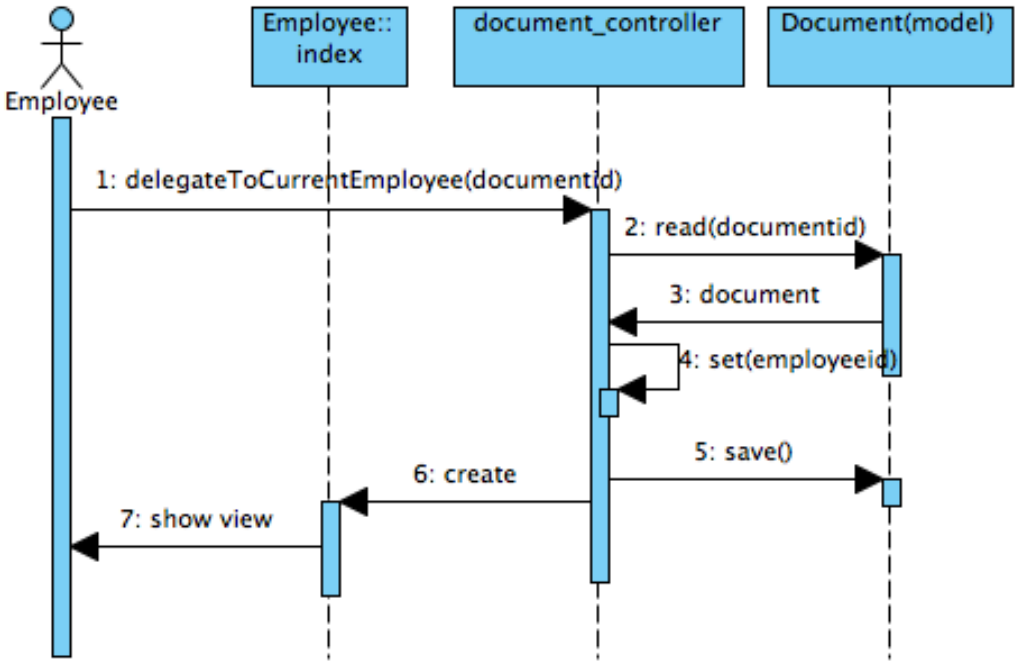


Figure 7.7: Employee Takes a New Document

Explanation for figure 7.7, described in use case 4.3:

1. The employee takes a document.
2. The document controller reads the document.
3. The model returns the document.
4. The employee id is set to the document.
5. The document is saved.
6. The index view is created.
7. The employee will see "The document was added to your tasks." in the index view.

The communications with the database are not shown in order to keep the readability of the diagram.

Changed implementations

The implementation for proofreading, shown in figure 6.6, has changed this sprint. We have added the possibility to undo your last upload. The new sequence diagram is shown in figure 7.8. Proofread is almost the same as translate, which also have the same changes. Therefore translate is not shown in the report. You can find it in figure D.1 in the appendix.

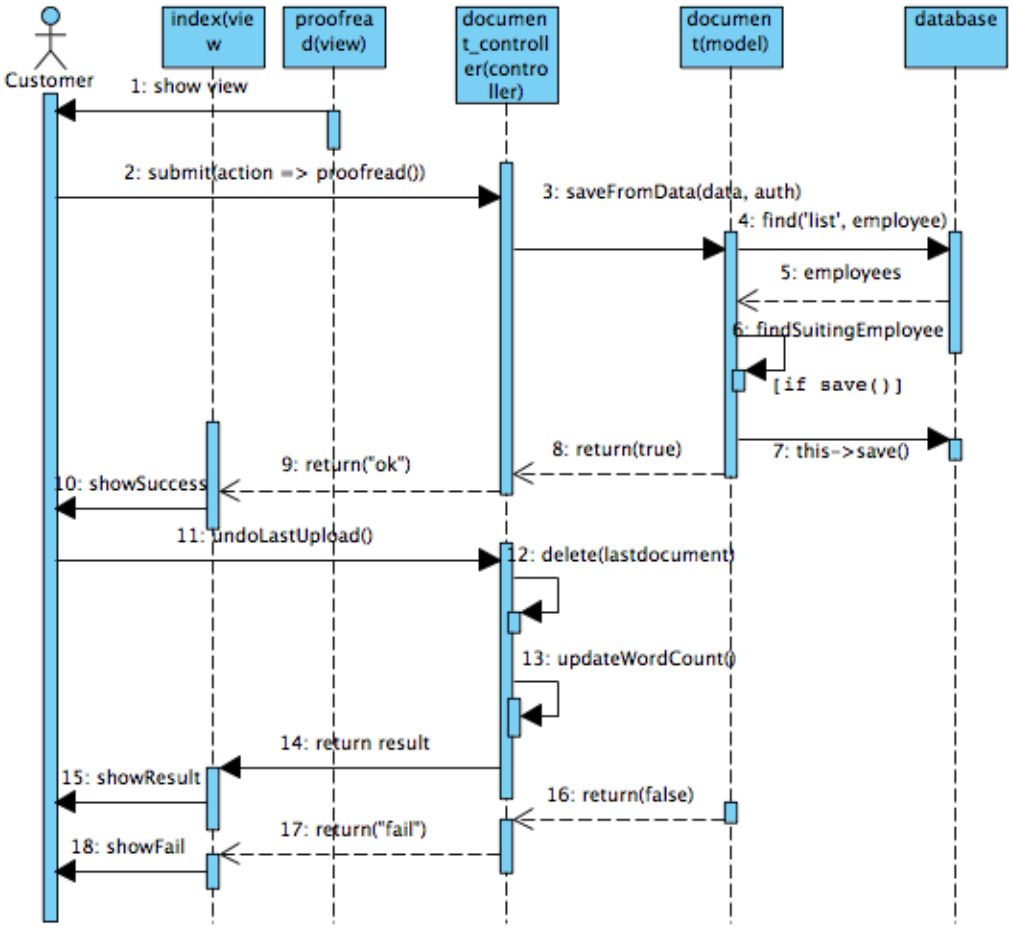


Figure 7.8: Uploading a Document for Proofreading

1. The proofread view is shown to the customer(the customer must have pressed the proofread link before. The data flow does not just start here.)
2. The customer inputs the informations he or she wants in the form and submits the form.
3. The customer controller lets the customer model do the logic on the data.
4. The customer model finds a list of all employees.
5. The database returns a list of all employees.
6. The model finds the most suiting employee and saves the it with the data.
7. If the data can be saved it is saved in the database(The else clause is removed to save some space. It is the same as in 6.6)
8. True is returned to the controller.
9. Create view with a message indicating that the upload was a success.
10. Show the view to the customer.
11. The customer can now press a link to undo the upload and then get back the used credit.
12. The last document uploaded to that user is deleted.
13. The credits are refunded.
14. The result view is created.
15. The result is shown to the customer.
16. If the data can not be saved false is returned to the controller.
17. Create a view with a message indicating that the upload was a failure.
18. Show the failure to the customer.

Item 2.9: The customer has the possibility to upload a document(See use case in figure 4.3), or to write a text in the form, in order to translate/proofread it. When a document is uploaded, there is a function that automatically assigns the document to an employee.

This assignment is not a random assignment, but it follows this logic.

The document is automatically assigned to an employee that has the right abilities (proofreading or translating) that the document needs, and that has the minimum number of tasks already assigned, in the group of the employees that has the abilities to handle this document.

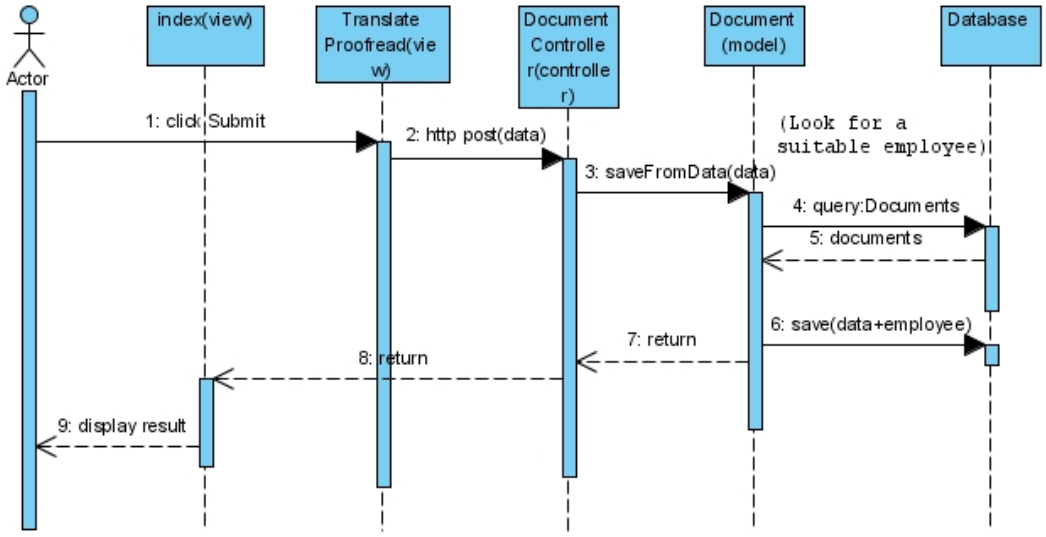


Figure 7.9: Document Automatically Assigned to an Employee

1. The customer press the submit button on the form to send the document for proofreading or translation. The information is then sent to the document controller.
2. The proofread/translate view sends the information from the form to the document controller.
3. The document controller sends the data from the view to the model for saving.
4. Before saving the document the model has to find an employee it can assign the document to. It asks the database for all employees, their qualifications and their current jobs. It will then find the best employee as mentioned above.
5. The database will return the employees to the model as an array.
6. If it is possible to save the document in the database it will be done.
7. The model return the value to the document controller.
8. The document controller redirects to the index view with a message.
9. The index view show a message to the customer.

Item 4.9: Word counting is one of the main features of the system. When a costumer writes a text, or submit a document, the words are counted in order to know if he has enough credit to apply for the task he has chosen. Otherwise the system will notify the costumer that he has to buy more words first of submit a document to be translated/proofread. Counting of the words is applied every time that a costumer writes or submit a new document, and it is also stored in the database with the other file information.

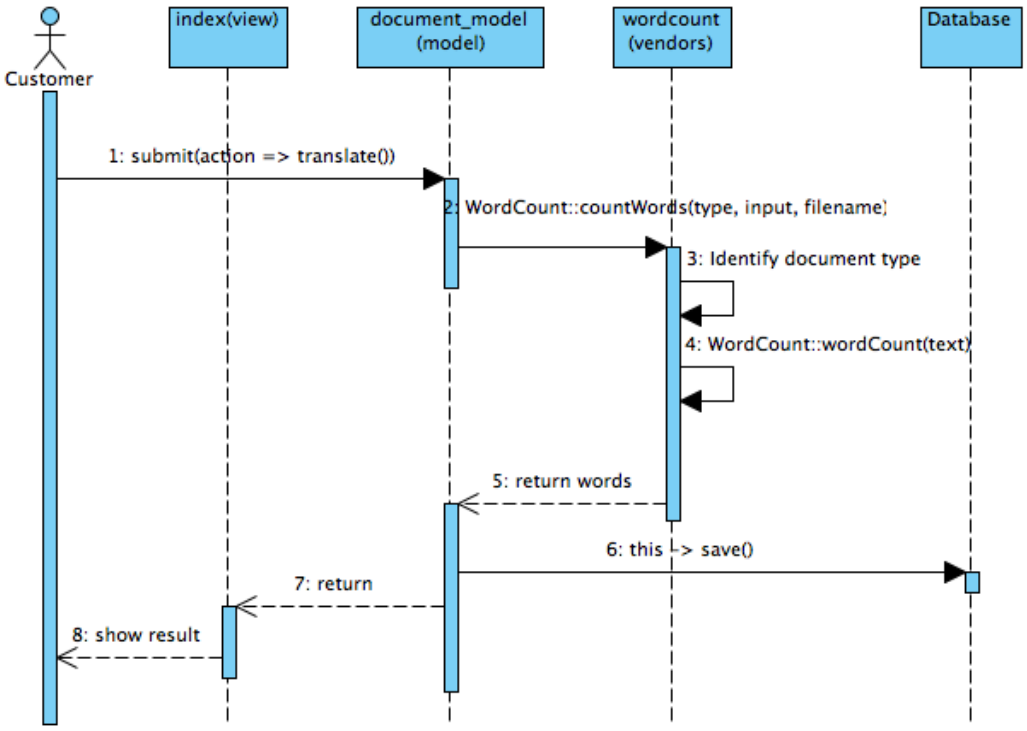


Figure 7.10: Words Counting

1. The Customer chooses the document that has to be uploaded, or writes a new text to be translated / proofread.
2. The Document model calls the word counting function in the word counting class.
3. The document type is identified in order to call the right function to extract the text.
4. Words are counted.
5. The number of words is returned to the document model.
6. The document is saved in the database and the right number of credits is taken away from the costumer's credits.
7. The confirmation is returned to the costumer.

7.2.2 Add-on deliverables

The main goal of this sprint for the add-on was to develop a portable version of the implemented features in the previous sprint. That means a runnable executable file has to be run by the final user which will fire an automatic installation of the implemented functionalities into Microsoft Word application on the user's machine. After having

had an in depth look on how that could be made and as it was first suggested by the customer in the planning sprint, the add-on application started to be implemented in C#. The Microsoft Visual Studio v9.0 environment, available on one of our computers, will from now on be used to develop this application.

Developing in C#

There are many different sort of things that can be done using C# and more generally with the associated .NET framework from Microsoft. Here the goal is to develop a software add-in for Microsoft Word. To do this you can either create a COM (Component object model) add-in, often called shared add-in, or a VSTO add-in (Visual Studio Tools for Office). It appears that VSTO add-ins solve many problems of compatibility and are are more suitable for the development of a Microsoft Office add-in for only one of its component (here the focus is on Word). COM add-ins are more often used when developing an add-in that aims to work for several Office applications (for example both for Powerpoint and Visio) [3]. So the choice we made is the development of a VSTO add-in.
The main difference when going from a simple VBA Macro to a full C# .NET project is that we are really experiencing an object-oriented software development. Indeed, we must always refer to the collection of classes we are using in the code, either on specifying the keyword `using` in the beginning of the code listing or always going all the way to the .NET collection of classes we want to use in the arborescence of the classes. For example, to use the `Range` object type of the `Document` collection, we must write: `Microsoft.Office.Word.Document.Range` when instantiating our object. Visual Studio is a huge help here, as it automatically adds needed collections when add-in common objects like forms and buttons.
Writing in C# is a bit different than writing in VBA. Visual Studio helps as it has auto-completion of statements and real-time error checking. At the same time it can be quite confusing, as functionality is split into namespaces and packages. The hard part is finding out which package and namespace the needed functionality is located in. Even though this makes it a lot more structured than programming in VBA, it can in some cases be more tricky to access the correct members and functions.

7.3 Sprint Testing

In were our test results are explained. See Appendix A.2 for the entire test descriptions.

Table 7.5: User Registration and Login Test Execution.

Title	User Registration and Login.
Test Identifier	SYWE-00, see table A.7.
Date	02.10.2009
Executioner	Riccardo Ortolan.
Test Result	User registration is one of the most important function of the system, and it has to be tested properly. Here it appears if a user finds any errors when he registers himself on the system, and an HTTP-400 response must always be granted, whether the registration form is well-compiled that it is not. We also check if some http or server errors are returned, failing the test if it happens. After the registration, we check if the user is able to log in into the system correctly, without any http errors, and handling eventual non-correct inputs. 1/1 test cases complete: 4 passes, 0 fails and 0 exceptions. The test is passed. The auth system of the PHP Framework is tested as well, in order to ensure that we will not have inconsistent data, and that we have a handled response in any case or combination of inputs. 1/1 test cases complete: 61 passes, 0 fails and 0 exceptions.

Table 7.6: Customer Account Settings Test Execution.

Title	Customer Account Settings.
Test Identifier	SYWE-01, see table A.8.
Date	02.10.2009
Executioner	Riccardo Ortolan.
Test Result	Users have the option to edit their personal information through a form. With this test we check that the user has a correct reply from the server, even if he submits wrong data. 1/1 test cases complete: 2 passes, 0 fails and 0 exceptions.

Table 7.7: Document Controller Function Test.

Title	Document Controller Function Test.
Test Identifier	UNTE-00, see table A.1.
Date	02.10.2009
Executioner	Riccardo Ortolan.

Test Result	The Document controller is tested in order to ensure that an user can submit a task without having any server errors, even if he submits improper data. 1/1 test cases complete: 2 passes, 0 fails and 0 exceptions.

7.4 Sprint Prototype

This section presents the finished prototype for this sprint. It is what we show the client at the review meeting. The section is separated in two parts, Web application and Word Add-in in C#.

Web application: The finished product for this sprint contained much more features than the last sprint. The employee user and administrator user pages where made so we had a lot more to show. One of the most important features is that the employee now can upload a respond to the customer. They will also automatically assigned to documents when they are uploaded. The customer user had also more features implemented. See the backlog 7.3.

Word Addin in C#: We started on this task and was hoping to have something to show, but the task of getting into C# and figuring out how to connect with Microsoft Word was a bigger task than expected. Therefore we had not much to show the client on this part.

7.5 Client Feedback

In the end of the sprint we had a small presentation of what we had done so far. The section is split into the new features of the web application, add-on in C# and new features for next meeting.

New features:

- Distribution of documents: Urgent translation/proofreading tasks should be passed on to an translator/proofreader as soon as possible and the translator/proofreader notified immediately. Limitations with regards to this need to be considered; Amounts of words that needs to be examined versus time, capabilities of the chosen person and manpower needed by LingTrans to be able to provide 24 hours service. The team should have in mind that Lingtrans must be able to provide quality; accurate and correct translation and proofreading.
- It should be possible to assign employees that are responsible for the urgent tasks; urgent task duty.

- In addition to built-in validation of input in CakePHP, we should implement our own validations as well.
- See if how CakePHP handles public/private fields/functions is sufficient.

Add-on in C#

- The client have nothing to add here. They were pleased with our presentation so far.

New features for next sprint: These are points that the group and the client together came up with as important points for next meeting.

- For the next sprint we should have a working version of the C# add-on.
- The administrator part of the web application should be overhauled.
- High-level design of a possible calendar for registering employee vacations and duty calls.
- The add-on API should be working before next meeting.
- Brainstorming results about the issue with the urgent translation and proofreading service. We talked about what would happen if a customer uploaded a document at the middle of the night, for example, and wanted a short deadline like 1 hour. Options are to always have some employee on watch or disable options that are not possible to fulfill.

7.6 Sprint Evaluation

In sprint 2 we had the problem of choosing to many backlog items which resulted in we had to work hard to make it. In this sprint we encountered another problem. The amount of items in the backlog was good, but some where still not implemented. The main problem was with the add-in implemented in C#. This turned out to be much harder than we first anticipated. Therefore this was not finished. See figure 7.11. The main problem was that we had not done a good enough preliminary study about the subject before starting. As a result a lot of the time was spent reading about C# and how to use this to make the add-in. Therefore, the implementation will continue in sprint 4.

The time log of each person should state what each group member have been doing. This has been done, but the entries was unspecific so it was difficult to know how much time used on each backlog item. The result of this is that a lot of time was spent on figuring out what item the time in the time log was actually used on. For the next sprint the time log entries will have to be more specific. Stating what item was worked on at any given time.

Delegation of work has improved in this sprint as team members have become more experienced with Scrum and the planning went better. This is a great improvement from sprint 2 were people had to wait for other members to finish their work before continuing. This could be explained by the fact that the core functionality of the system has been implemented and that the team members roles are more relaxed. Each members skill set is now better understood by the group.

For the next sprint we will continue as we did this sprint with the planning and delegation of tasks. The main problem this sprint was with the add-in in C#. There have just been one person working on this and as the main functions of the web application is done we will assign more manpower to the add-in.

Burndown Chart : The burndown chart 7.11 illustrates the work left in the sprint with respect to the estimated hours.

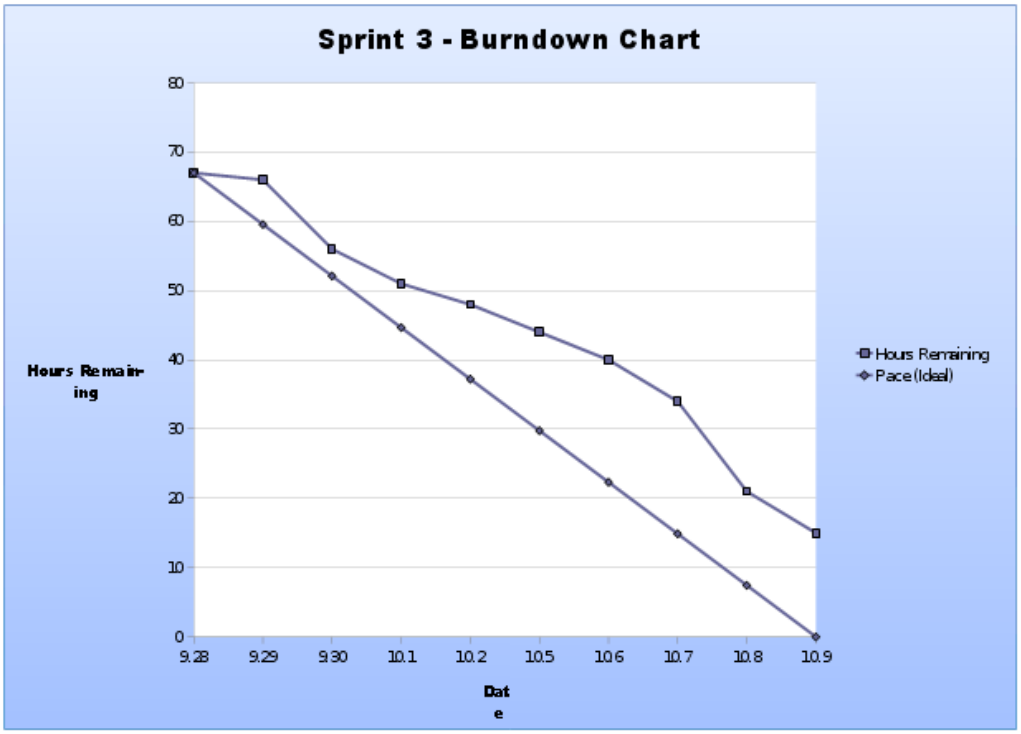


Figure 7.11: Burndown Chart Sprint 3

CHAPTER 8

Sprint 4

All the most important features are now implemented. In this sprint we will continue to implement some medium priority items, but also implement some low priority items from the product backlog.

This chapter presents the goals, backlog and delivered features.

8.1 Sprint Planning

In this sprint we will continue as before, but with one major change. Since all of the most important features on the web application are implemented and the add-in proved to be a bigger task than we first thought we will focus more on the add-in by using more people on that task.

8.1.1 Overall Sprint Goal

The web application will get less focus in this sprint than before and the add-in will get more. We will in this sprint glue the web application together and polish what has already been made. In addition we will have a functioning add-in by the end of this sprint. The system focus for this sprint is illustrated in the High Level System Focus Diagram 8.1 below.

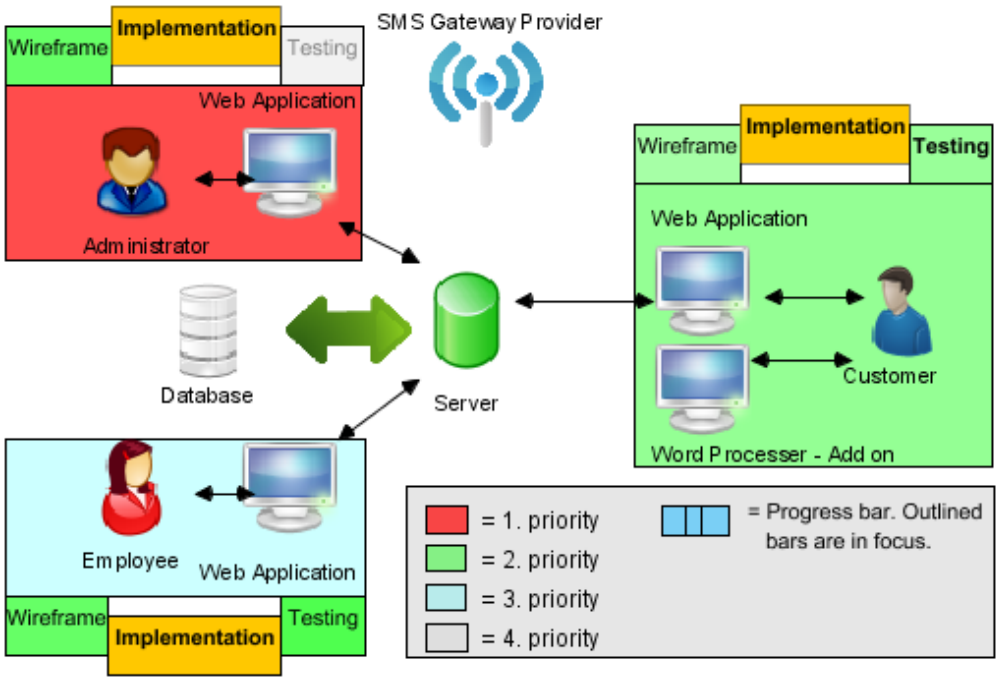


Figure 8.1: High Level System Focus Diagram

8.1.2 Sprint Backlog

These are the backlog items from this sprint. There are some high, medium and low priority items.

Table 8.1: Web Application Backlog - Sprint 4

Item	*Priority*	*Task*	*EH*	*AH*
Customers				
1.12	H	When a customer buys more credits, this should be entered in the billing database	2	2.
2.17	M	A customer should be able to see pricing options on the home page, without login in.	2	2
3.1	L	A customer should be able to contact the administrators through the website.	2	2
3.3	L	A customer should be able to choose if he or she wants notification by email when a task is finished.	1	1
Employees				
2.7	M	An employee should be able to set his or her availability when logged in.	16	12

Continued on next page

Item	*Priority*	*Task*	*EH*	*AH*
Administrators				
2.8	M	An administrator should be able to set the skills for the employees, when logged in	3	3
2.18	M	An administrator should be able to see all sent and unsent bills in the database when logged in.	2	3
2.16	M	An administrator should be able to see a log of all recently performed tasks on the website.	8	6
3.4	L	An administrator should be able to add new administrators to the system when logged in.	2	2
3.5	L	Administrators will be able to register new employees in the system.	1	1
System				
5.1	H	The system should report to the administrators if the available employees do not cover supported languages.	1	2
TotalHours			40	36

Table 8.2: Add-on Backlog - Sprint 4

Item	*Priority*	*Task*	*EH*	*AH*
Microsoft Word Add-on				
4.8	M	The customer should be able to download an installer file to install the MS Word add-on.	6	3
4.10	L	The MS Word add-on should be ported to C#.	16	16
OpenOffice Add-on				
4.9	L	The add-on should be ported to work in the latest version of OpenOffice Writer.	-	-
TotalHours			22	19

8.2 Deliverables

There has been added some models to the application since sprint 2. Localization and LocalizationLanguage has been removed from the diagram in this sprint. It is still in the

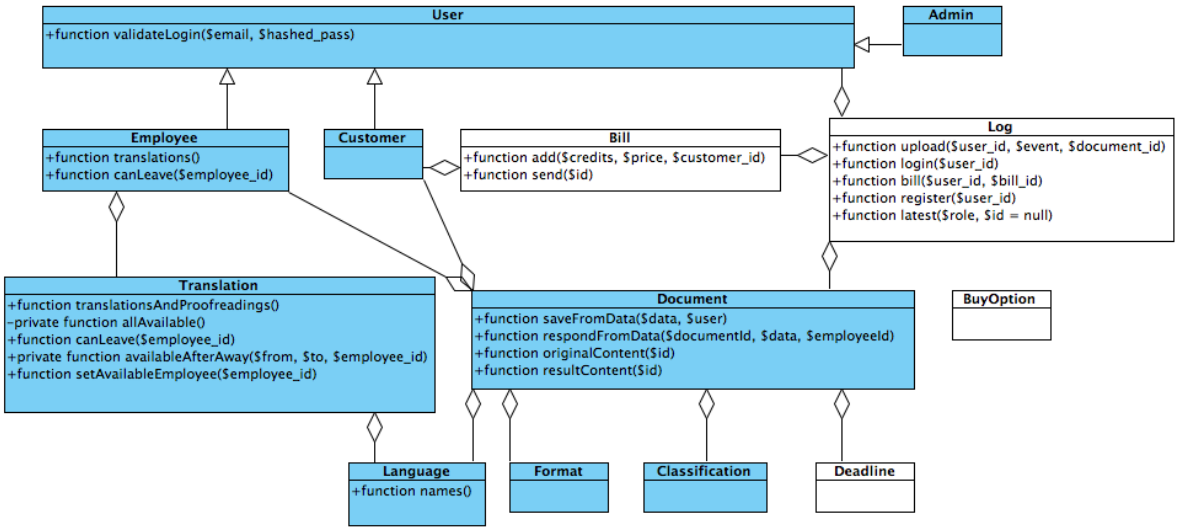


Figure 8.2: MVC Models Classes

code, but is not used. These model was used to save all the text in different language on the page in the database. We have moved away from this idea and just saved all text in an array in the code. Figure 8.2 is an updated version of figure 6.5. Figure 8.2 do also include the functions that each model includes. Some of the models correspond to the database tables, while others do not. See ER-diagram 8.3.

Table 8.3: Added Models

Model	Description
Bill	This holds the bills for the customers. It also does some logic with the bills.
Log	This holds the log.
BuyOption	Used for the different options the customer have when buying more words.
Deadline	Holds the different deadlines for.

The database has also been reworked. See the old ER-diagram in figure 5.3. Figure 8.3 shows the new database. It now has 17 tables, some of which has been added because of new requirements from the client: the bills and logs tables are new features not planned from the beginning, also a buy_options table has been added in order to obtain all the possible kinds of prices and combinations. Localization tables have been created in order handle the different types of languages included in the website. Some minor fixes have been done in order to fit the design issues encountered during the development of the PHP Framework.

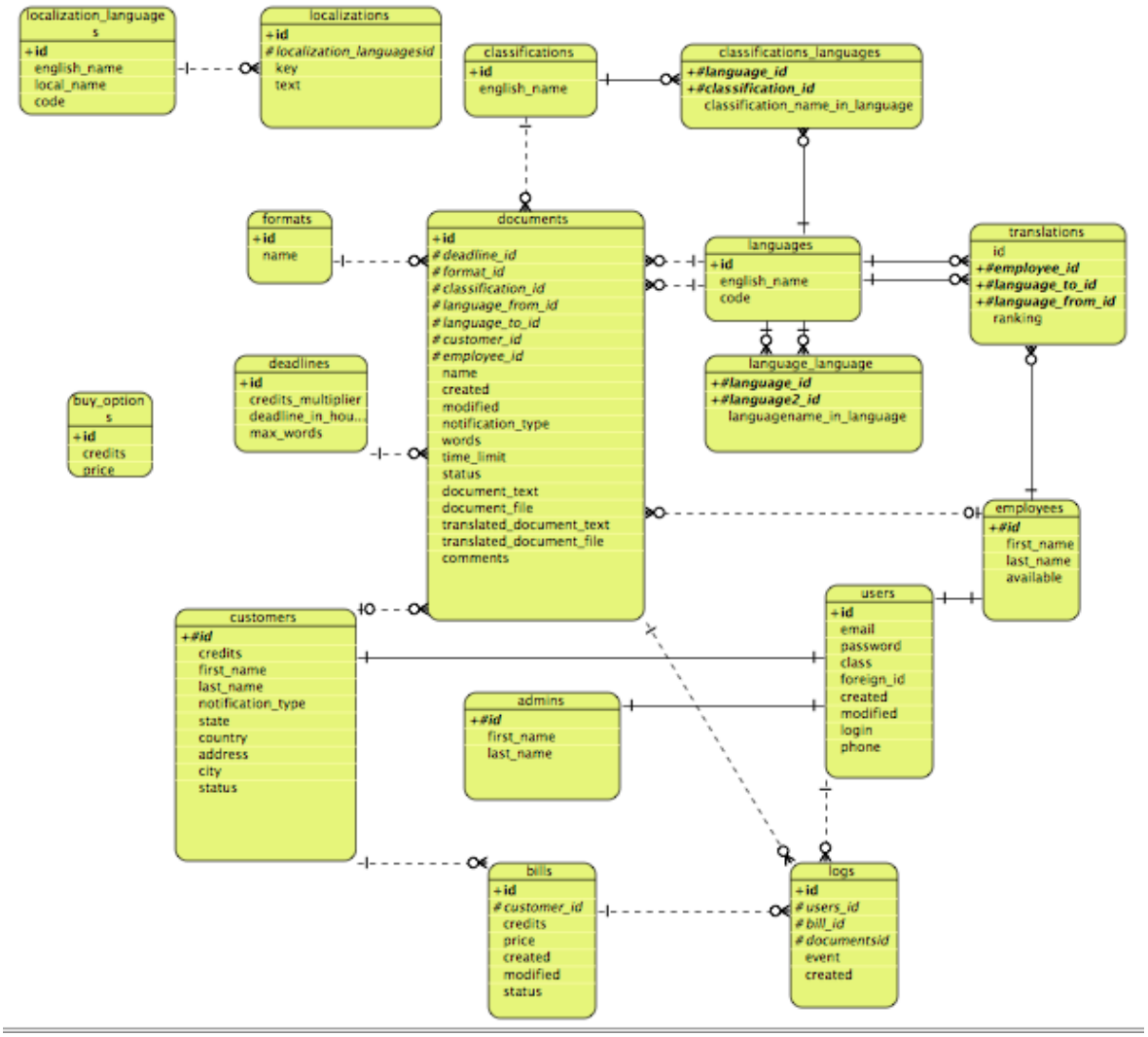


Figure 8.3: ER-diagram

8.2.1 Web Application Deliverables

Item 1.12: A customer can buy credits to have them available for when he or she is going to upload a document. For us this is just for saving the information. Lingit will later have to implement their, already made, system with ours. Figure 8.4 is the sequence diagram for this. The communications with the database is not showed in order to keep the readability of the diagram.

Explanation for figure 8.4:

1. The customer press the buy link in the buy view and the customer controller function buy() is called.

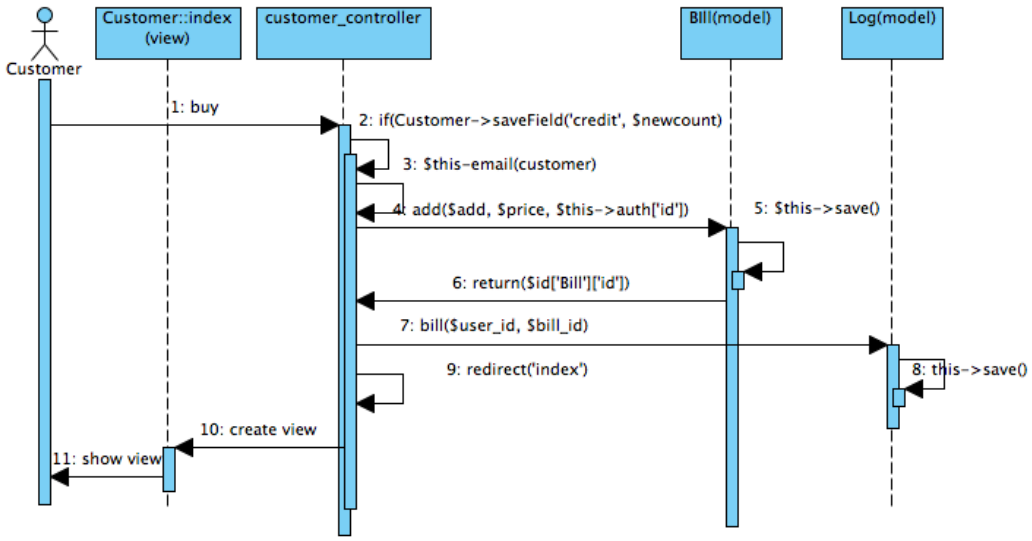


Figure 8.4: Enter Billing Information in the Database

2. If it is possible to save the new credits count in the credits field on the customer the function will continue.
3. The customer is emailed with the billing details.
4. The new billing information is sent to the bill model with the add function.
5. The bill model saves the billing information.
6. The id of the bill is returned the the customer controller.
7. The customer controller calls the bill function in the log model to save the bill as a log entry.
8. The log saves the logs.
9. The customer controller redirects to the index view.
10. The index view is created.
11. The index is shown to the customer with a message showing the success of the billing info.

Item 2.17: The customer is able to see the pricing for the service without logging in. The view `pages/index.ctp` will when created look in the database and list all the prices available.

Item 3.1: This item is listed for the customer, but it is also available for the employee. If the User has any problems he should contact the administrator. At the bottom of each page there is a link the user can click. The sequence diagram in figure 8.5 is the sequence diagram for the use case in figure 4.3. The communications with the model and the database are not shown in order to keep the readability of the diagram.

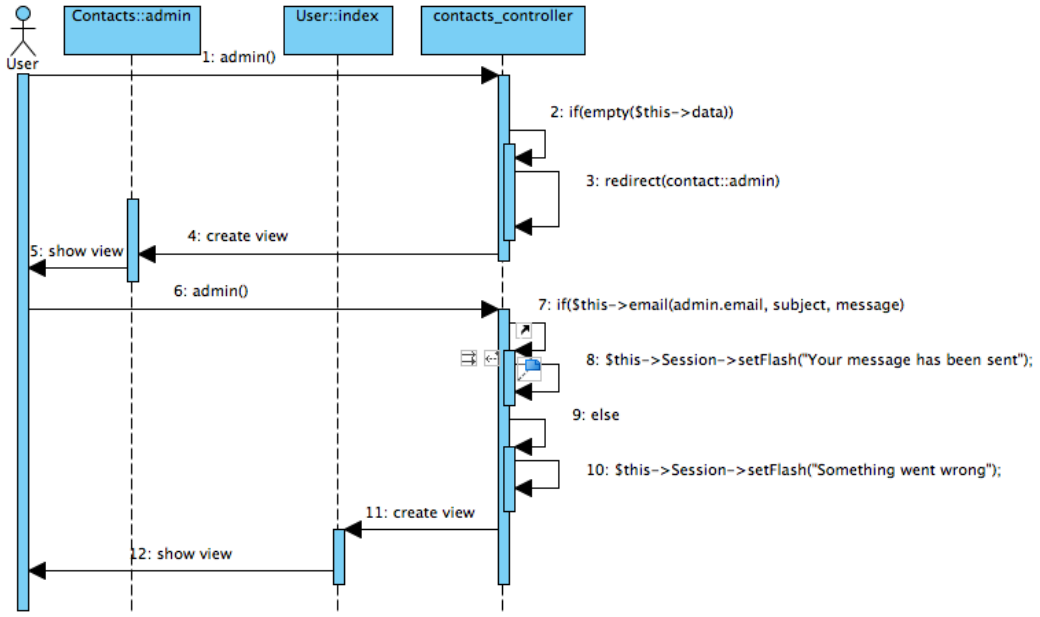


Figure 8.5: Contact Admin

Explanation for figure 8.5:

1. The user press the "Contact administrator"-link. It is available in all pages because the link is in the default layout.
2. The contact controller checks that the data is empty.
3. The user is redirected to the Contacts:admin view.
4. The admin view is created.
5. The admin view is shown to the customer. This contains a form where the user can insert his message and subject to the admin.
6. The user has inserted his message and press submit. This calls the admin function in the contacts controller.
7. The data is now not empty and the the function tries to send the message to the admin.
8. If this succeed a message of the success will be set.
9. If the function can not send the email for any reason another part of the code will be initialized.
10. A message of the failure will be set.
11. The index view of the user who contacted the admin will be created with the set message.

12. The view will be shown to the user with the message.

Item 3.3: The customer should be able to select what kind of notification he wants. The customer will have to press "Account settings" from for example the customer index view. Figure 8.6 is the sequence diagram for this and it corresponds to the use case in figure 4.3. The communications with the model and the database are not shown in order to keep the readability of the diagram.

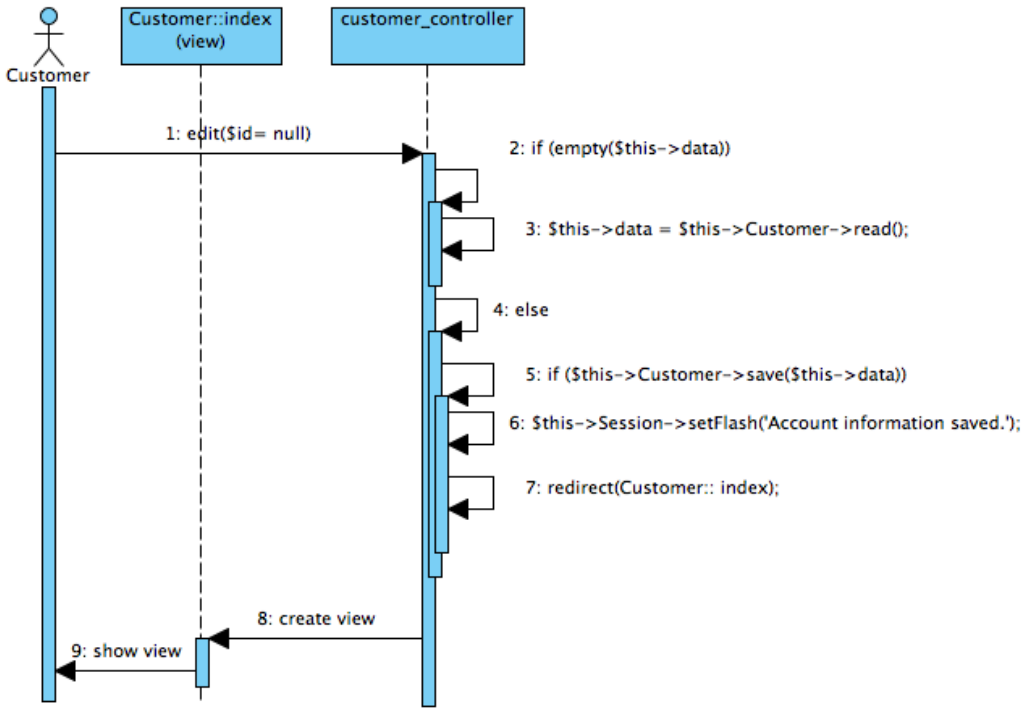


Figure 8.6: Edit Notification

Explanation for figure 8.6:

1. The customer press the "Account Settings"-link. This will call the function edit in customer controller.
2. If the data is empty it means the customer have not pressed submit. The customer will here see his/her information in a form which he/she can edit. Not shown in the diagram.
3. Read in the customer information.
4. If the data is not empty it means that the user has pressed the submit button on the edit view.
5. If the new information can be saved the function proceeds.
6. A message is set about the success.

7. The customer is redirected to the index view.
8. The index view is created with the message.
9. The index view is shown to the customer with the message.

Item 2.8: The administrator can set the skills of each employee. The administrator have the ability to list all the employees. From there he can edit each one of them. This is done by adding them in a form. Figure 8.7 shows the sequence diagram for this. The communications with the model and the database are not shown in order to keep the readability of the diagram.

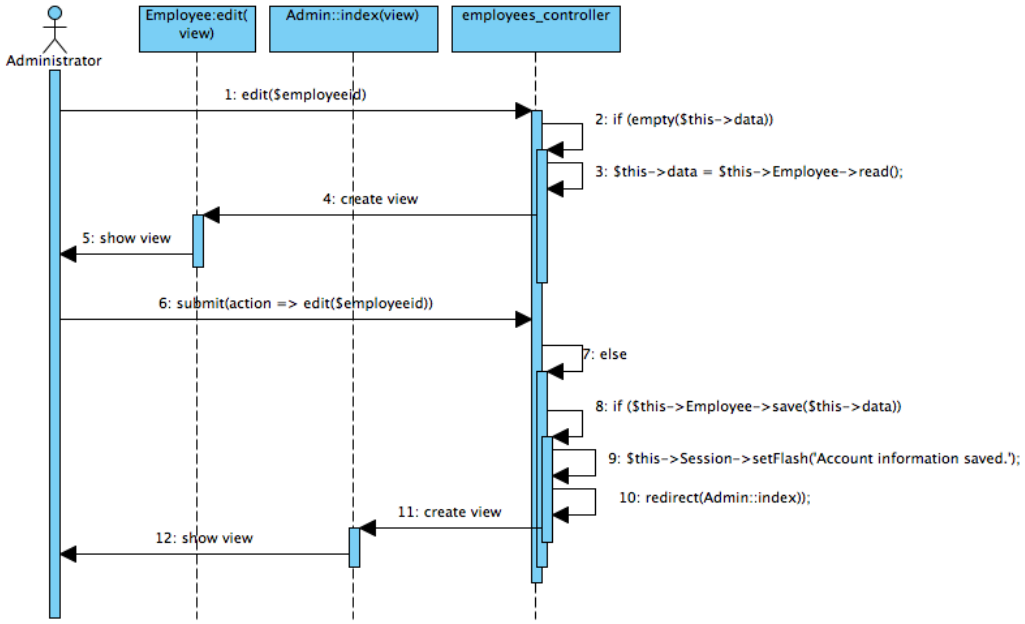


Figure 8.7: Set Skills for Employees

Explanation for figure 8.7:

1. The administrator will be looking at the list of employees and from there he can press edit.
2. The administrator has not added any data yet.
3. The chosen employee is read into the model.
4. The Employee:edit view is created showing all the information about the chosen employee.
5. The view is shown to the admin.
6. The administrator will add the skill of the employee and submit.
7. Now the data is set so the else clause kicks in.

8. If the data can be saved the function proceeds.
9. A message is set of the success.
10. The admin is redirected to the index view.
11. The index view is created.
12. The index is shown to the customer.

Item 2.7: The employee can set the status to not available. The purpose of this is to inform the system that this person is away, thus removing deadline and language options from the service if necessary. If the employee is the only one that can proofread in italian, this option can no longer be chosen by the customer. The status change is done by pressing the "unavailable" button on the employee page. Figure 8.8 shows the sequence diagram for this. The communications with the model and the database are not shown in order to keep the readability of the diagram.

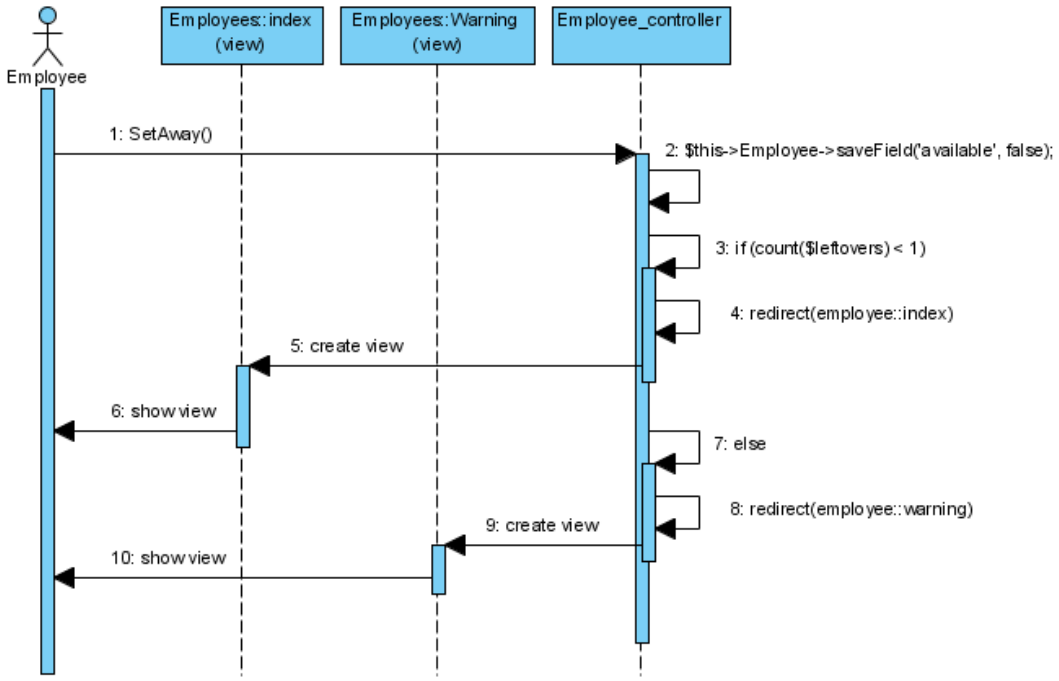


Figure 8.8: Set Employee Status to Unavailable

Explanation for figure 8.8:

1. The employee presses the unavailable button. It is available in all pages because the link is in the default layout.
2. If the employee is registered as available in the database, proceed. .
3. The controller is called with the set away function.

4. The controller checks to see if the system has to disable any languages because of this employee.
5. If that is not so, the employee is registered in the database as away and that no languages had to be disabled.
6. The employee is redirected to the employees index view.
7. The index page is created and shows the updated default view and confirms that the employee is successfully listed as away.
8. If any languages had to be deactivated, this procedure commence:
9. Create message to the employee informing about the situation.
10. Redirect the employee to a warning view.
11. Create the warning view and display the warning message for the employee.

Item 2.18: The view bills/index.ctp will query the database and create the list of unsent and sent bills and display it to the administrator.
Item 2.16: When the administrator logs in the index view will be loaded. This is done by calling the function index in the admin controller. This function sets a variable in the view. The variable will contain the 50 latest entries in the log. This is done by calling the function latest in the log model which query the database for the latest created logs. The log items themselves are inserted by various functions in the application for example when the customer registers. The customer controller calls the function register in the log model which again add the log entry in the database.
Item 3.4 and Item 3.5: An administrator can list all the administrators and from here he can add new administrators. Figure 8.9 shows this in detail. This corresponds to the use case in figure 4.3. The communications with the model and the database are not shown in order to keep the readability of the diagram. The administrator can also add employees, but since this is a really similar task to adding administrators it is not shown here.

Explanation for figure 8.9:

1. The administrator will click on a button to register new administrator. This calls the register function in the admin controller.
2. The controller creates the register view which contains a form to add new administrators.
3. The view is shown to the Administrator.
4. The administrator have inserted the information and submits the new administrator information.
5. In this call to the register function the data is not empty so the data is now saved and the administrator is redirected to the admin::all view.

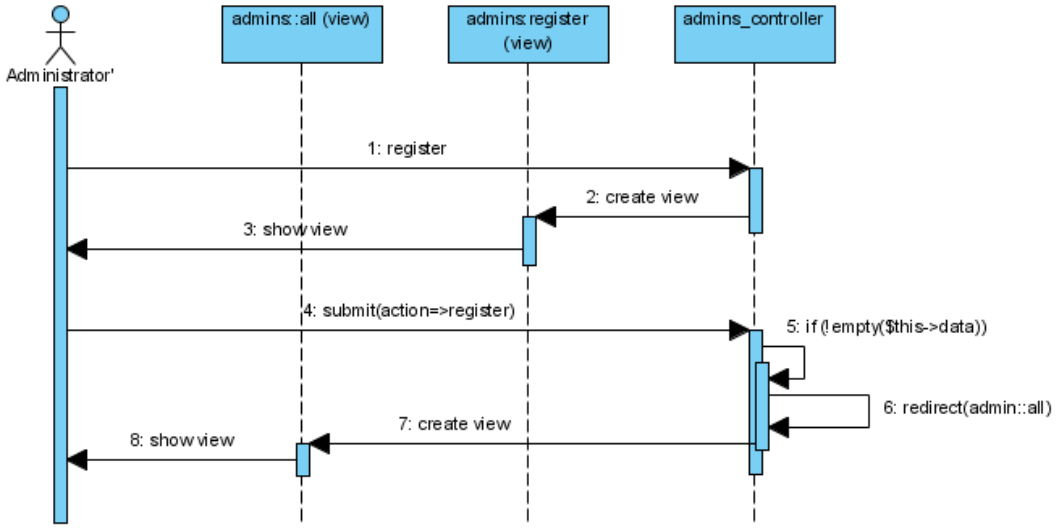


Figure 8.9: Add Administrator

6. The view is created with the new administrator in the list.
7. The view is shown to the administrator.

8.2.2 Microsoft Word Add-in Deliverables

Item 4.10: A big amount of hours has been spent in the previous sprint to learn how C# works, and especially the VSTO tool of Visual Studio for developing Microsoft Office. In this Sprint, we went from the macro developed in Sprint 2 into an executable file that can be installed on the system.
The code is now protected and the add-in is known as an additional program in the control panel of Windows, so it can also be easily removed from the final user's machine. The use cases has been slightly modified (see figure 8.10) compared to what was done in the macro. The reason for this was that we simplified the way methods are called and above we made the add-in more user-friendly. Indeed, now he/she has the possibility to check news without any concerns about whether the user has selected a text in the application or not, which was useless before.

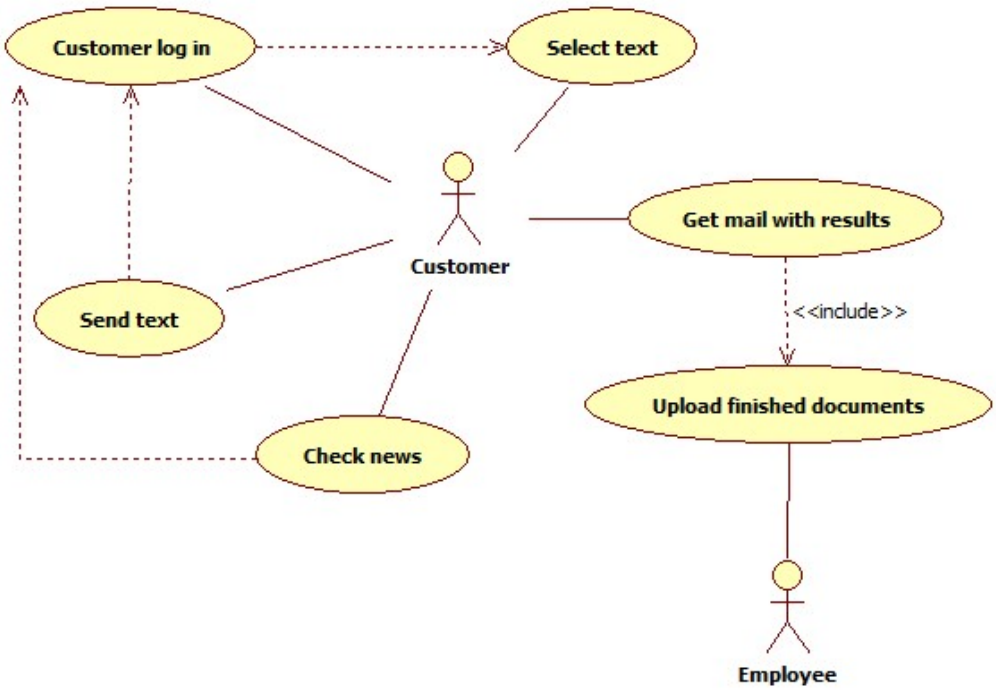


Figure 8.10: Microsoft Word Add-in Use Cases

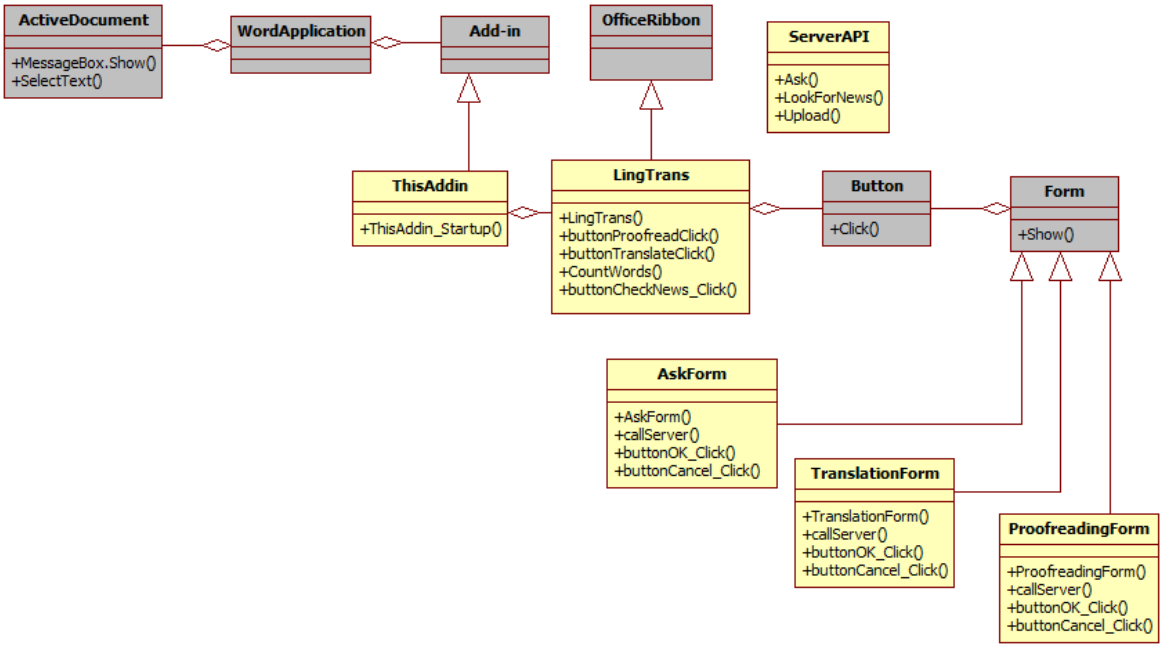


Figure 8.11: Microsoft Word Add-in Class Diagram

Figure 8.11 shows a class diagram for the Microsoft Word add-in. In gray are shown the already existing classes from the Microsoft package, such as the forms or the Ribbon for example. The other classes are some that we created to stick to the needs of the application. From the Windows classes, we only listed the classes we use in the application, as the list can grow really huge and therefore compromise the readability of the diagram. In table 8.4, a description of all the classes is shown.

Table 8.4: Word add-in classes description

Class	Description
ThisAddin	Object representing the current Add-in. This is the class that creates all the controls available to the user
LingTrans	It is the ribbon shown in Office, in which all the buttons available to the user are actually visible. It inherits from the OfficeRibbon class because it contains methods that are not in this mother class.
AskForm	A type of Windows Form that contains additional methods to interact with the server. Created in the 'CheckNews' use case.
TranslationForm	A type of Windows Form that contains additional methods to interact with the server. Created in the 'Translate' use case.
ProofreadingForm	A type of Windows Form that contains additional methods to interact with the server. Created in the 'Proofread' use case.
ServerAPI	The API used to interact with the server. All the web requests are made to this class, which sends back the informations to the add-in and more (it also sends e-mails to users).

The uses cases has been modified to better suit the users' needs. Here are the sequence diagrams of the use cases shown above in the new use case diagram. Table 8.5 and figure 8.12 refer to the *Check News* use case, table 8.6 and figure 8.13 refer to the *Translate* use case and Table 8.7 refer to the *Proofread* use case

Table 8.5: CheckNews Description

Name: Check News	**Objectives**: Connect the customer from Word to see if there are some new documents ready for him.

Primary actors: A customer **Secondary actors**: None	**Dates**: 26.10.09 **Responsible**: Brian Mathé **Version**: Sprint 4
Preconditions: The customer is already registered (in the database) and not connected.	**Postconditions**: The customer is told wether there are some new documents ready for him or not.
Scenarii: When clicking on the 'Check news' button, the user is asked his or her username and password in a windows form. An HTTP request is sent to the server containing the user's data in an XML string to the server API. If a new document is available, the server sends back in an XML file if some news documents are ready for him or not. This XML is parsed and shown in a message box to display the informations to the user.	

Use case: Check news

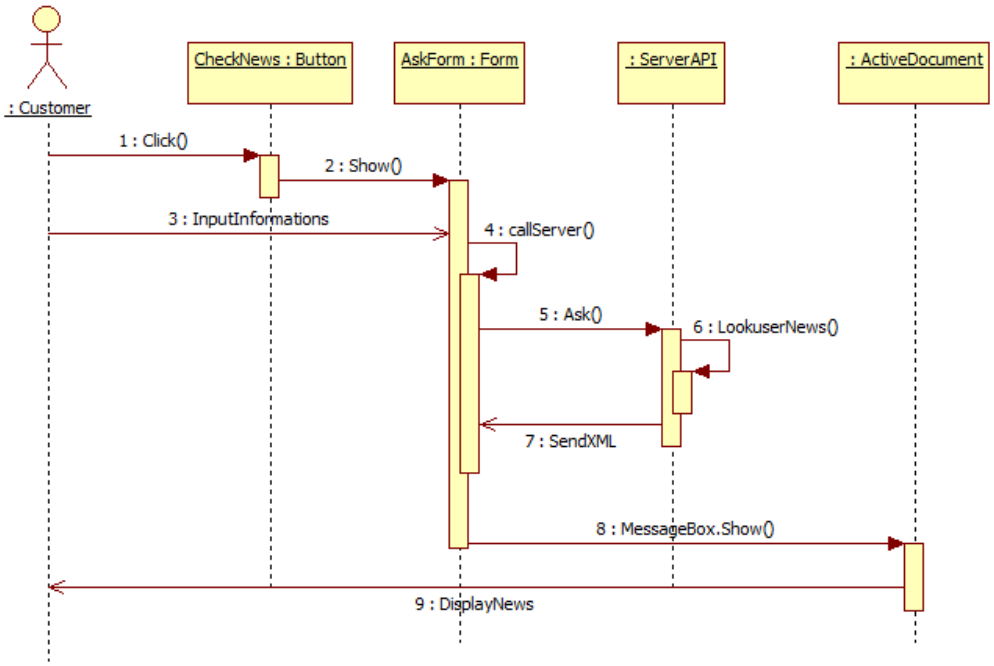


Figure 8.12: CheckNews Use Case Sequence Diagram

1. The user clicks on the 'Check news' button of the ribbon.
2. The AskForm is displayed as a windows form.

3. The user types its username and password.
4. When pressing the OK button in word, it fires the `callServer()` method in which an XML string containing the user informations is created and a web request to the server is made.
5. Within the `callServer()` method, the `ask()` method is distantly called on the server API.
6. A specific method of the API is called to check if the user has new documents ready.
7. The API sends back the informations in an XML string.
8. A message box is created in the active document to display these informations.
9. The user can see if there are some new documents ready for him.

Use case: Translate

Table 8.6: Translate Description

Name: Translate	**Objectives**: Send the text to translate to the web service.
Primary actors: A customer **Secondary actors**: An employee	**Dates**: 26.10.09 **Responsible**: Brian Mathé **Version**: Sprint 4
Preconditions: The customer is already registered (in the database) and not connected.	**Postconditions**: The text to be translated is sent to the server and saved in the database.
Scenarii: When clicking on the 'Translate' button, the user is asked his or her username, password , the language he or she is translating from, the one he or she wants the text back in and the deadline. If no selection has been made, a message box advise the user to select the text he or she wants to send. An HTTP request is sent to the server containing the user's data in an XML string to the server API. The text is then saved in the database and a new job is created in the selected language. When the employee sends back the document, an e-mail is sent to the user.	

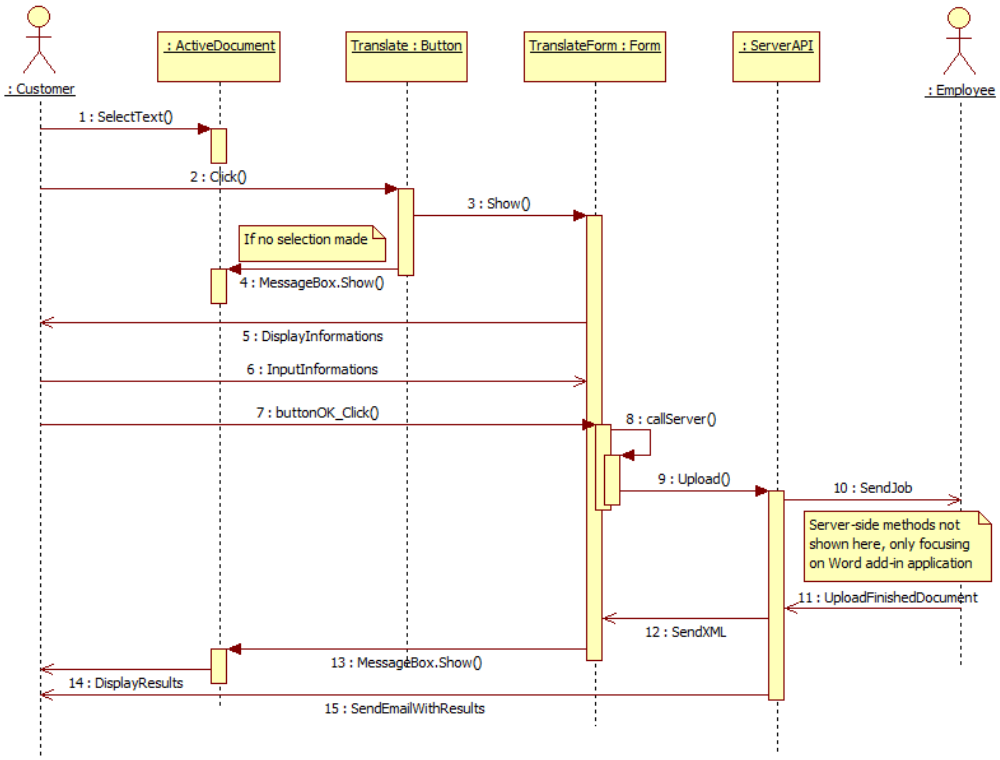


Figure 8.13: Translate Use Case Sequence Diagram

1. The user selects the text he/she wants to send for translation.
2. The user clicks on the 'Translate' button
3. The TranslationForm is displayed as a windows form.
4. If no selection has been made by the user, he is advised to make his selection with a message box.
5. The user can see the length of his/her selected text and can see the form to input informations.
6. The user types its username, password, the language he's writing in, the language he wants the output document from and the deadline.
7. The user presses the OK button.
8. When pressing the OK button in word, it fires the `callServer()` method in which an XML string containing the user informations is created and a web request to the server is made.
9. Within the `callServer()` method, the `upload()` method is distantly called on the server API.
10. A new task is created in the database.

11. The employee uploads the finished document.
12. An XML string is sent to the add-in when checking for news (see previous use case) to inform the user new documents are available.
13. A message box is displayed in the active document to inform the user wether there are some new documents or not.
14. The user can see if there are some new documents available.
15. An email is sent to the user to notify him/her of new documents available.

Use case: Proofread

Table 8.7: Proofread Description

Name: Proofread	**Objectives**: Send the text to translate to the web service.
Primary actors: A customer **Secondary actors**: An employee	**Dates**: 26.10.09 **Responsible**: Brian Mathé **Version**: Sprint 4
Preconditions: The customer is already registered (in the database) and not connected.	**Postconditions**: The text to be translated is sent to the server and saved in the database.
Scenarii: When clicking on the 'Translate' button, the user is asked his or her username, password, the language he or she is using and the deadline. If no selection has been made, a message box advise the user to select the text he or she wants to send. An HTTP request is sent to the server containing the user's data in an XML string to the server API. The text is then saved in the database and a new job is created in the selected language. When the employee sends back the document, an e-mail is sent to the user.	

The sequence diagram is exactly the same as the *Translate* use case but the only difference is that a different type of form is created at the operation number 3. Moreover, there's only 1 language (the one of the document being used) sent to the server, even though we call the same method `upload()`.

8.2.3 OpenOffice Writer Add-on Deliverables

In addition to the Microsoft Word add-in one of the goals for this sprint was to port the add-on to OpenOffice. The only two available languages for developping an OpenOffice

extension where the code is unreachable are Java and C++. C++ was quickly disregarded as a choice because of the excessive amount of time it would take. The language has no built-in libraries to take care of the network communication between the extension and the server, which would be a whole project in itself. Java was therefore our only real choice.

Sun has developed an extension for NetBeans to create shell code for OpenOffice add-ons. This extension sets up everything needed to load the add-on into OpenOffice and create menu buttons and so on. The problem we met here was that it was not compatible with the latest version of NetBeans. The solution was therefore to downgrade to the previous version, but this proved to be incompatible with the latest version of OpenOffice. After a lot of trying it actually proved to be impossible to find a suitable combo of the different versions of the programs. There also exists a similar extension to Eclipse - another JAVA IDE - which was the next logic choice. This also proved to be incompatible with the latest version of Eclipse, and was in addition only in the alpha stage of development.

The only choice left was to develop it in a scripting language, which would make it a macro instead of an extension. This means that the code would be fully visible to the user, and with limited functionality. It would basically end up being the same as our Visual Basic version of the Microsoft Word add-on. We discussed our findings with the client, and together concluded to skip the development of this extension, and instead focus more on the add-on for Microsoft Word and finalization of the web application.

8.3 Sprint Testing

In were our test results are explained. See Appendix A.2 for the entire test descriptions.

Table 8.8: Email Notification Function Test Execution.

Title	Email Notification Function Test.
Test Identifier	UNTE-01, table A.2.
Date	02.10.2009
Executioner	Erik Andreas Larsen.
Test Result	With this test we ensure that the functionality of the email sender is tested, in order to know if an user receives an email when his document is ready, and if an employee receives an email when there is a new task for him. All the functionalities are working and the emails are received correctly by the addresses.

Table 8.9: Employee Availability Test Execution.

Title	Employee Availability Test.
Test Identifier	SYWE-03, table A.10.
Date	02.10.2009
Executioner	Erik Andreas Larsen.
Test Result	This test ensures that an employee is able to set the available status to away or to available. There is no need to add input data or form checking. The system responds correctly to the employee status changing.

8.4 Sprint Prototype

This section presents the finished prototype for this sprint. It is what we show the client at the review meeting. The section is separated in two parts, Web application and Microsoft Word Add-in in C#.

Web application: We are now almost finished with the web application, so this sprint we can show almost all of the features to the client. The most important one is the administrator part. He can now add new translators and proofreaders and set their skills. Most of the page have gotten some kind of rework to make it look and/or function better.

Microsoft Word add-in: The add-in is functional for this sprint. It can upload documents to the server and can be installed with an own windows installer. This will automatically connect it to word so users can use it.

OpenOffice Writer add-on: The add-on was not started on, after some consulting with the client. Because of the difficulties with the implementation, and to ensure the quality of the two other products, we together agreed to forget about this add-on.

8.5 Client Feedback:

Given the next sprint is the last one for implementation, every missing feature has to be done in two weeks. Because there are maybe a lot of things to do, an additional meeting will take place next week at the middle of last sprint, in order to see how things are going and maybe redefine priorities of things to do. The client feedback on our presentation is separated into three parts: Web application, Microsoft Word add-in and missing features.

Web application

- When registering one could not register with two first names. The application did not accept a space in the name. This has to be fixed.
- The calculation of deadlines has to be given in working hours, and that has to be explicit for the customers. It has to take care of the holidays. We maybe should find a web service that checks if a certain date is a holiday or not.
- Localization: we need to localize the entire web page. NB/NN is ISO-standard.

Microsoft Word add-in

- Fix the various bugs appeared during the presentation. In particular, we should detect if the user is connected to the internet.
- When there are new documents, there should be a link to the web application displayed when the final user presses the 'Check news' button.
- It would be nice for the user if we could remember his or her username and password. Only the username should be remembered by the system, to prevent security issues.
- The add-in has to automatically check for news when logging in. The will require the add-in to remember the username and password or at least the username.
- It would be nice if the add-in automatically set the default language of the 'language from' field to the language being used in the document being written.
- The add-in should get the available languages automatically from the server.

Missing features

- SMS/Gateway: Make a dummy function that simulates sending a sms when needed.
- We have to make the video presentation for the web application and embed it on the web page. This only has to be done in english for now.

8.6 Sprint Evaluation

In this sprint we did not encounter any problems with regards to the number of items we planned to do, as we have gained experience with the Scrum method. We did not miss any feature to be implemented. We also did not find any problems with the new features and changes that the client ask us to implement after the meetings, as we do not have any more important things to learn about the PHP framework, and we spent a lot of less time to implement them. We showed a demonstration of the add-in to the costumer and we chose which items was the most important ones to finish. We also discovered some minor bugs that needed fixing. Like for example that you could not register with two first names. This was easily fixed.

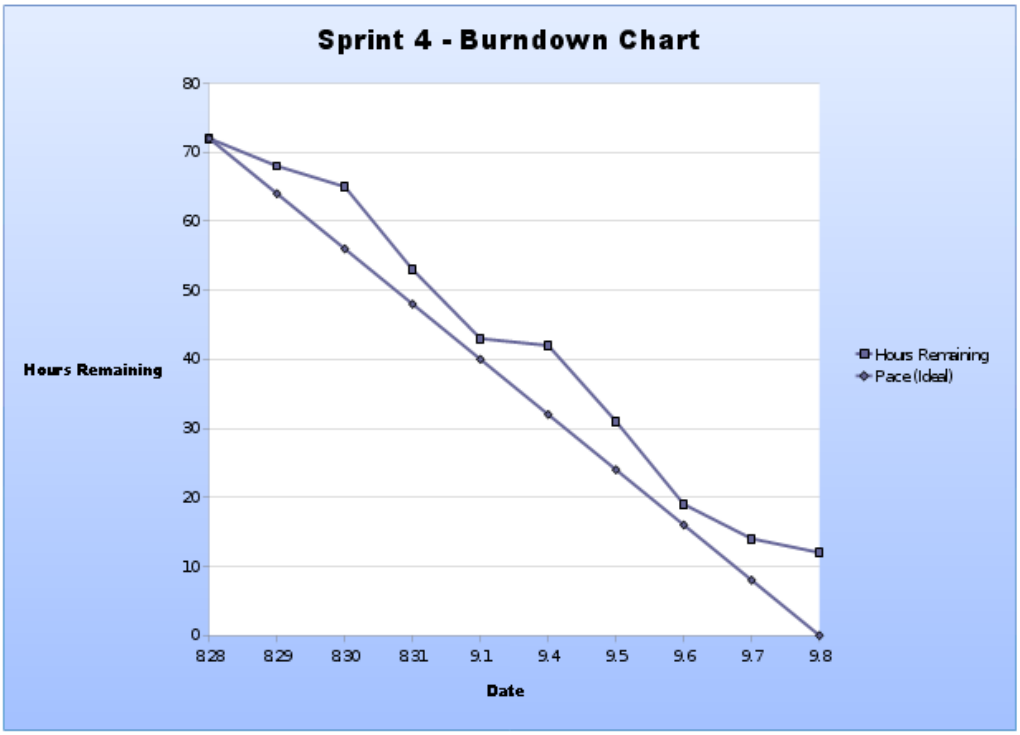


Figure 8.14: Burndown Chart Sprint 4

Burndown Chart : Figure 8.14 illustrates the work left in the sprint with respect to the estimated hours. The reason why the line do not reach zero remaining hours is because item 4.9, "The add-on should be ported to work in the latest version of OpenOffice Writer" was not implemented, see section 8.2.3.

CHAPTER 9

Sprint 5

Sprint 5 is the final sprint of implementation. All of the items in the web application backlog have been implemented, with the exception of item 3.2, so this sprint will mainly consist of finishing touches and finalizing our application. There will also be some implementation on the Microsoft Word add-in to make it more complete and better working.

This chapter presents the goal, backlog, implemented features and evaluation of sprint 5.

9.1 Sprint Planning

This section will explain the overall goal of this sprint, and present the reader with the backlog.

9.1.1 Overall Sprint Goal

The overall goal of this sprint is to finalize and polish the web application as much as possible, and to implement as much of the remaining features on the add-in as possible. We will also focus a lot on testing implemented features to assure a certain quality in the final product. The client has stated that they also want to test the application, so we have agreed to focus only on comments from the client during the last week of this sprint.

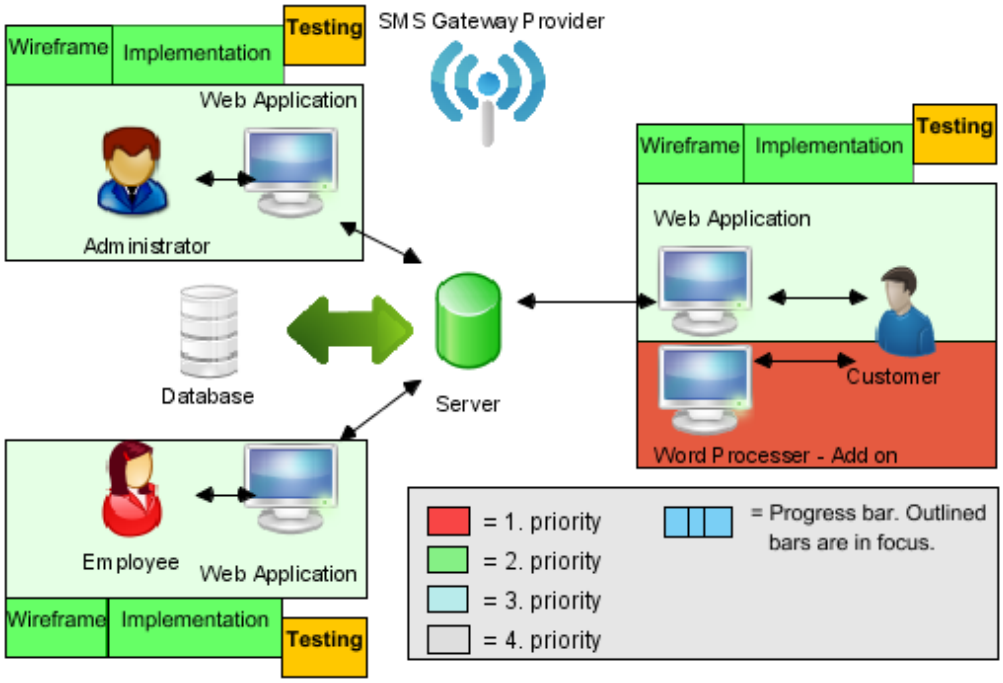


Figure 9.1: High Level System Focus Diagram

9.1.2 Changes to the backlog

After the presentation of the add-in at the end of last sprint the client had some comments which we feel should be added in the backlog. All these are sub tasks of porting the Microsoft Word Add-in so they have the item number 4.9.X where 4.9 is the item of porting. The added item can be found in table 9.2. The complete product backlog for the add-in can be found in table 4.4.

9.1.3 Sprint Backlog

The sprint backlog for the web application is quite short in this sprint. However, in this sprint there will also be a lot of focus on bug fixing, refactoring code, testing code and in general finalizing the system for delivery. These tasks may not have items in the feature backlog, but are very important nonetheless.

Table 9.1: Web Application Backlog - Sprint 5

Item	*Priority*	*Task*	*EH*	*AH*
2.19	M	An employee should receive mail about new available tasks	2	2
Continued on next page				

Item	*Priority*	*Task*	*EH*	*AH*
3.2	H	A customer should have the option to get an SMS notification when a translation or proofreading task is finished	2	2
TotalHours			4	4

Table 9.2: Add-on Backlog - Sprint 5

Item	*Priority*	*Task*	*EH*	*AH*
4.9.1	H	The add-in should detect if the user is connected to the internet	2	2
4.9.2	M	The add-in should be able to have a check news button when there are new documents, there should be a link to the web application displayed when the final user presses that button	1	3
4.9.3	M	The add-in should remember the username and the password of the customer in the current session	3	2
4.9.4	M	The add-in should remember the username of the customer when Microsoft Word is closed and then started again	3	3
4.9.5	M	The add-in should automatically set the default language of the language from field to the language being used in the document being written	3	3
4.9.6	M	The add-in should automatically get the available languages and the deadlines from the server	4	4
TotalHours			16	17

9.2 Deliverables

This section contains the description of all the deliverables for this sprint. That includes both the web application and the Microsoft Word add-in.

9.2.1 Web Application deliverables

Item 3.2:
After consultation with the client, it was decided that finalization of the SMS features would be left out of this sprint. This was decided since the client themselves had to make arrangements with a third party SMS gateway provider, and since there already was a lot of finalization to do in this sprint.

The SMS features was made ready for such a provider by creating all the calls to a SMS library found in `app/vendors/sms.php`. All that remains is to make arrangements with a gateway provider, and use their API to implement the `send` function in the SMS library

9.2.2 Microsoft Word Add-in Deliverables

Item 4.9.1:
This item came as we experienced some bugs on the demonstration during the previous client meeting. When a user tries to use some functions of the add-in in Word, he or she needs to be connected to the internet. We implemented a function that tells the user he orshe is unable to use the service while not connected to the internet. This detection is made in doing a ping to the API.

Item 4.9.2:
Instead of just saying to the user that there are some documents ready for him, we added a link to the web application page, where the user can log in to see his or her new documents.

Item 4.9.3:
We found it pretty annoying to have to type the username and password within the same session in Word every time a new form is opened, and it was also a remark coming from Lingit to implement a function that remembers the user's informations. So we made a function in the code that remembers the latest typed username and password within a session. When starting a new session, those fields are empty in the forms. This is an important point because it means that there is nothing saved outside of the built-in code in a separate file for example. So there should be no security issues with that function.

Item 4.9.6:
When the user want to translate or proofread a document the add-in have to get the available languages and deadlines from the server. A customer can not upload documents in languages not available at Lingit at the given moment. The deadlines may change so it will be better to just change them in the database and let everything else retrieve the information from there.

When 'Proofreading' is pressed the server will, at launch of the form, ask the server for the languages and deadlines. This happens with exchange of information with XML text sent through HTTP requests.

The same goes for translation, but then the customer choses a language to translate from the form will ask the server for what languages are available to translate to from that specific language. An updated sequence diagram of the use case *Translate* is shown in figure 9.2. Exactly the sale procedure is happening for the Proofread use case without the test for translation availability in operation number 10.

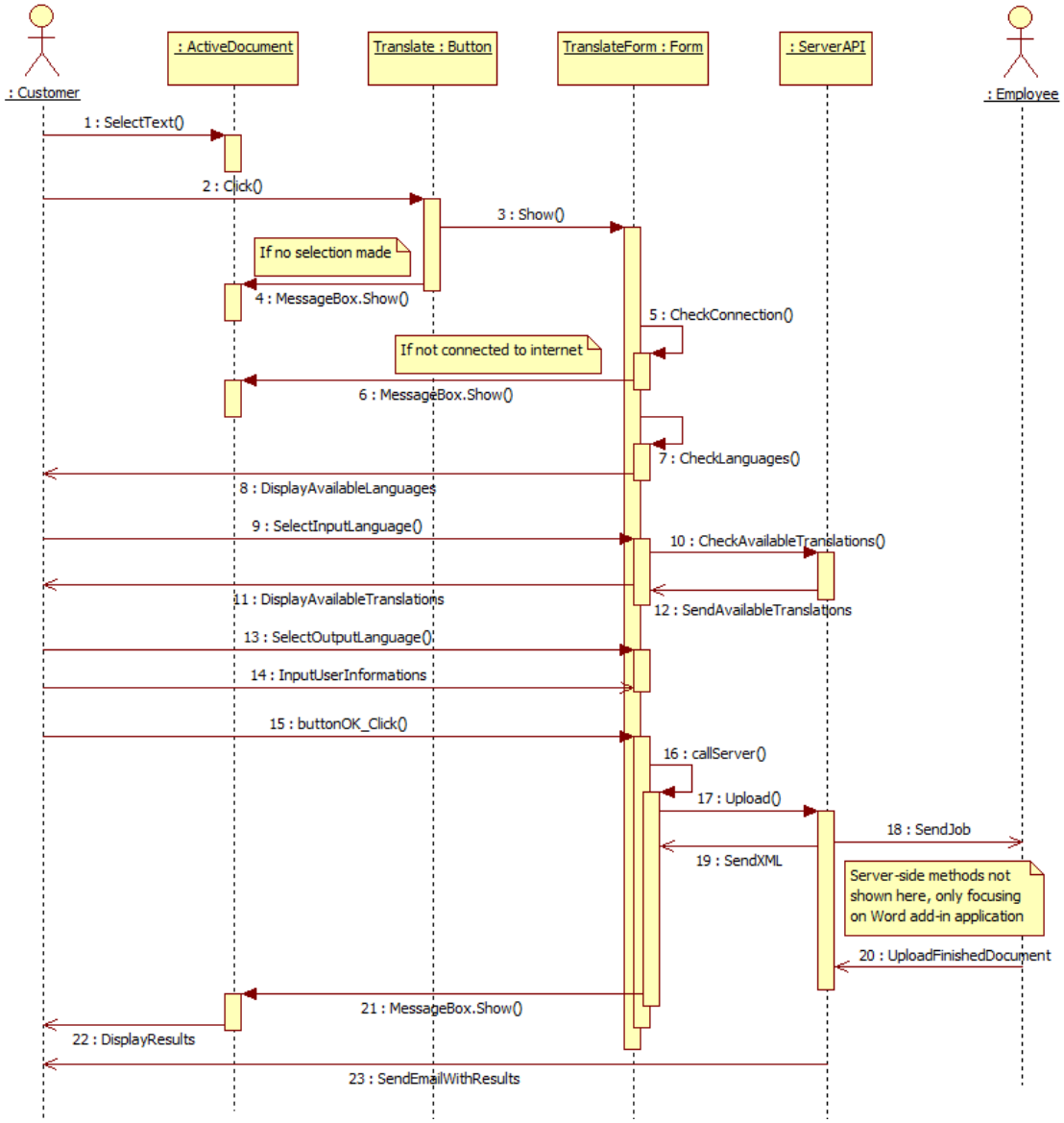


Figure 9.2: Translate Sequence Diagram for Add-in

9.3 Sprint Testing

This section contains the tests for this sprint. See Appendix A.2 for the entire test descriptions.

Table 9.3: Administrator Management Test Execution.

Title	Administrator Management.
Test Identifier	SYWE-02 , table A.9.
Date	02.10.2009
Executioner	Erik Andreas Larsen.
Test Result	An administrator is able to create and edit a new employee from his page, filling up the form, and after deleting an user, another admin or an employee it will not be displayed in the list anymore. An administrator is able to look at any finished and unfinished documents, and to look at any languages supported by the system. By deleting one of them it will not be shown on the supported languages any more. An administrator is able to see all the logs of the activities. An administrator is able to look at every sent and unsent bills, and he is able to mark any bill as sent. By doing this it will be showed in the sent bills area. Test Passed.

Table 9.4: Performance Test Execution.

Title	Performance Test.
Test Identifier	PETE-00, tabe A.11.
Date	02.10.2009
Executioner	Erik Andreas Larsen.
Test Result	Testing was sucessful. Each page has a loading time below 0,5 seconds. The loading time for each view is listed below, first and second review, respectively. **Independent of role** front-page: 0.104s, 0.12s, register: 0.108s, login: 0.1s, about:0.356s, disclaimer: 0.14s **CUSTOMER role** welcome-page: 0.144s, 0.128s, customers: 0.1s, 0.132s, translate: 0.132s, 0.128s, proofread: 0.14s, 0.128s, edit: 0.172s, 0.116s, buy: 0.12s, 0.108s **EMPLOYEE role** employees: 0.16s, 0.132s, contact admin: 0.184s, 0.132s, repond to task: 0.168s, 0.144s, set away: 0.144s, 0.12s

	ADMIN role admins: 0.248s, 0.156s, all customers: 0.148s, 0.172s, all employee: 0.192s, register new employee: 0.14s, 0.136s, all admin: 0.136, 0.116s, register new admin: 0.148s, 0.136s, all domcuments: 0.292s, 0.208s, languages: 0.136s, 0.112s, bills: 0.164s, 0.12s

Table 9.5: Customer Usability Test Execution.

Title	Customer Usability Test.
Test Identifier	USTE-00, tabe A.12.
Date	02.10.2009
Executioner	Erik Andreas Larsen.
Test Result	Passed. The persons used to play the role as customer had to rank themselves as a novice, advance or expert user, depending on their general internet usage. Novice user, test person one: used 4 minutes blank. Advance user, test person two: used 3 minutes and 36 seconds. Expert user, test person three: used 1 minutes and 25 seconds.

Table 9.6: Framework Controller Tests.

Title	Framework Controller Tests.
Test Identifier	UNTE-02, table A.3.
Date	05.10.2009
Executioner	Riccardo Ortolan.
Test Result	All the Framework controllers have been tested. 9/9 test cases complete: 507 passes, 4 fails and 0 exceptions. 4 nonsignificant tests fail in the Email controller due to different body field in the testing email and due to deprecated sendemail commands.

Table 9.7: Framework Models Tests.

Title	Framework Models Tests.
Test Identifier	UNTE-03, table A.4.
Date	05.10.2009
Executioner	Riccardo Ortolan.

Test Result	All the Framework models have been tested. 8/8 test cases complete: 934 passes, 0 fails and 0 exceptions.

Table 9.8: Framework Helpers Tests.

Title	Framework Helpers Tests.
Test Identifier	UNTE-04, table A.5.
Date	05.10.2009
Executioner	Riccardo Ortolan.
Test Result	All the Framework helpers have been tested. 17/17 test cases complete: 1997 passes, 2 fails and 0 exceptions. 2 nonsignificant tests fail in the HtmlHelper due to deprecated html tags.

Table 9.9: Framework Database Tests.

Title	Framework Database Tests.
Test Identifier	UNTE-05, table A.6.
Date	05.10.2009
Executioner	Riccardo Ortolan.
Test Result	All the Framework Datasources, Schema and DbAcl have been tested. 3/3 test cases complete: 501 passes, 3 fails and 0 exceptions. 3 nonsignificant tests fail due to deprecated mysql commands.

Table 9.10: Customer Acceptance Test Execution.

Title	Client Acceptance Test.
Test Identifier	ACTE-00, table A.13.
Date	02.10.2009
Executioner	Erik Andreas Larsen.
Test Result	Passed.

9.4 Sprint Prototype

This section presents the finished prototype for this sprint. It is what we show the client at the review meeting. The section is separated in two parts, Web application and Microsoft Word Add-in in C#.

Web Application
The web application is as finished as it will be in this project now. Read more about the end product in the next chapter. The next chapter also contains a section about what would be the next step in developing this product.

Microsoft Word add-in
The add-in is now fully functional. It can send documents for both proofreading and translation, and it gets the available languages and deadlines from the server. After the last client meeting we found some features that was not included in the backlog for the add-in. Almost all those were implemented in this sprint. See table 9.2.

9.5 Client Feedback

This section contains the client feedback on our presentation.

General Comments:

- Lingit will run a test on the system. We will fix small bugs that they find. Bigger problem will not be fixed, but rather documented.
- As a general statement, good work has been done and good solutions has been found to the problems encountered, sometimes better than expected. Though, like every IT project, a few things are missing which will be carefully documented.

Web Application

- A user manual for the customers, employees and administrator have to be made.
- Holidays should not be included when counting the working hours left on a certain task. This is now done by adding dates to a array. This can easily be extended.
- The virus checking is made as stub function. Lingit will later fill these when they start using the system.
- Since we can not use private/public with cake, we have to document this.
- The document size should not be restricted by mediumblob, which is 16Mb. This was fixed during the meeting. It is now a longblob, which is 4Gb.

Microsoft Word Add-in

- The system should check for available languages again when the user press submit of a document. Now the system just checks for availability when the user opens the dialog. This is because something may have changed since the user opened the dialog box.
- We have to document that we do not save the username between sessions.
- The check news feature of the add-in did not work properly. This have to be fixed.
- We have to document what system we have made and tested the add-in for.

9.6 Sprint Evaluation

Because the web application in essence was finished during the last sprint, this sprint was mainly about fixing bugs and removing unwanted and unintended functionality. Therefore, there have been no bigger changes to the web application. We have been able to test the system as a whole and see that it all works the way it is supposed to. Finding and eliminating bugs can be seen as a challenging task for a numerous reasons, but the use of CakePHP test suite made it a lot easier. We still had to find the source of our bugs ourselves, but knowing where they should be makes it a lot easier.

After the client meeting in the end of sprint 4, we agreed that there were some features that needed to be implemented in The Microsoft Word add-in. Most of these were features that would make the add-in more streamlined and easier to use and some functionality to make it more integrated with the web application. These features were changes that would not be visible to the end user, but necessary nevertheless. There were no bigger problems implementing them.

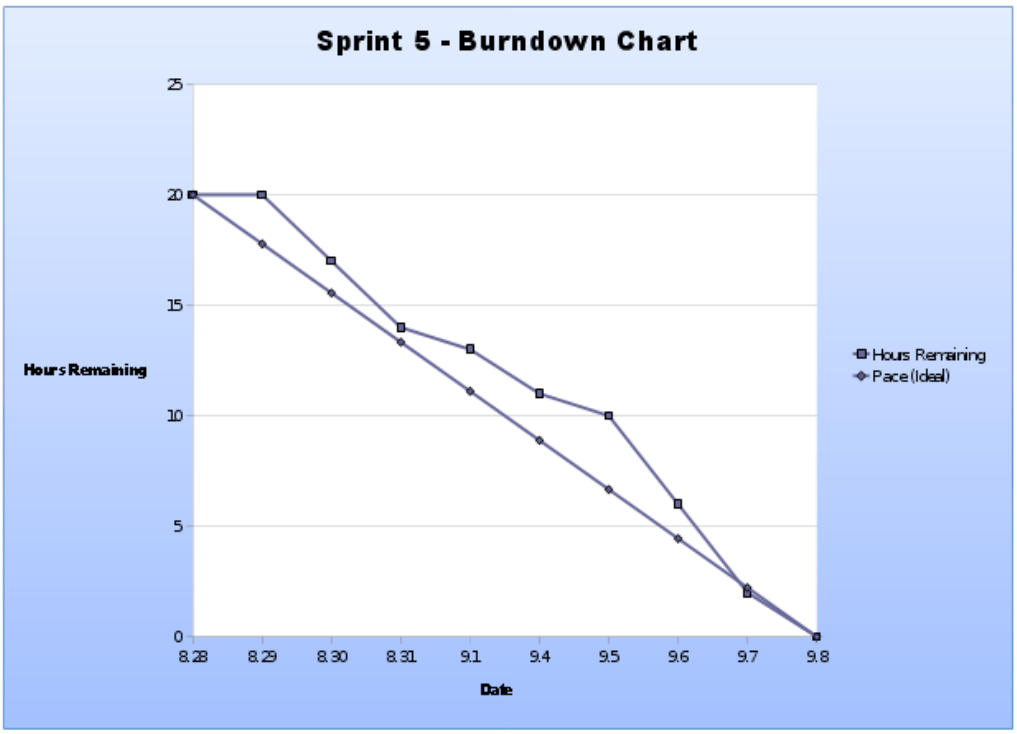


Figure 9.3: Burndown Chart Sprint 5

Burndown Chart : The chart 9.3 illustrates the work left in the sprint with respect to the estimated hours. It shows that week two of the sprint was the most efficient one with regards to implementing. Week one was mostly used to find bugs and running system tests.

Part III

Conclusion & Evaluation

Introduction

Purpose

The purpose of this part is to conclude the project and evaluate the development process. It will give an overview of the final system and sum up all the sprints.

Scope

Overall System Architecture explains the system architecture using various diagrams, discusses several issues for further development and improvement of the system, and states the systems current limitations such as programming language and file format support. **Project Evaluation** discusses the development process in terms of group dynamics, the choice of tools and technology, risks and time usage.

CHAPTER 10

Overall System Architecture

This chapter contain wrap up information about the system as a whole. It explains how it works in a very general and overall way. It also explains the current limitations of the system, and possible future changes.

10.1 Overall Architecture

This section will try to give a very overall description of the architecture of both the web application and the Microsoft Word add-in. Figure 10.1 shows the overall function of the system. It shows how the Microsoft Word add-in, the web application and the database is connected.

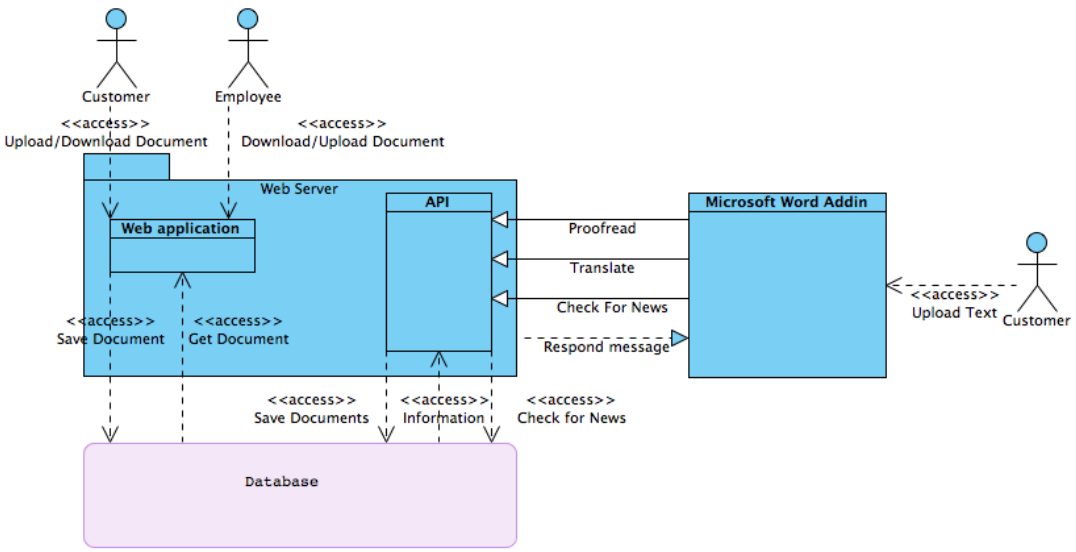


Figure 10.1: Overall Function of the system

10.1.1 ER Diagram

A reworked final version of the database has been created, the localizations and localization_languages tables have been deleted because they were unnecessary in order to localize the pages. Table language_language, built to obtain the name of the languages in their respective language is deleted aswell, since that data is obtained from a file. See figure 10.2

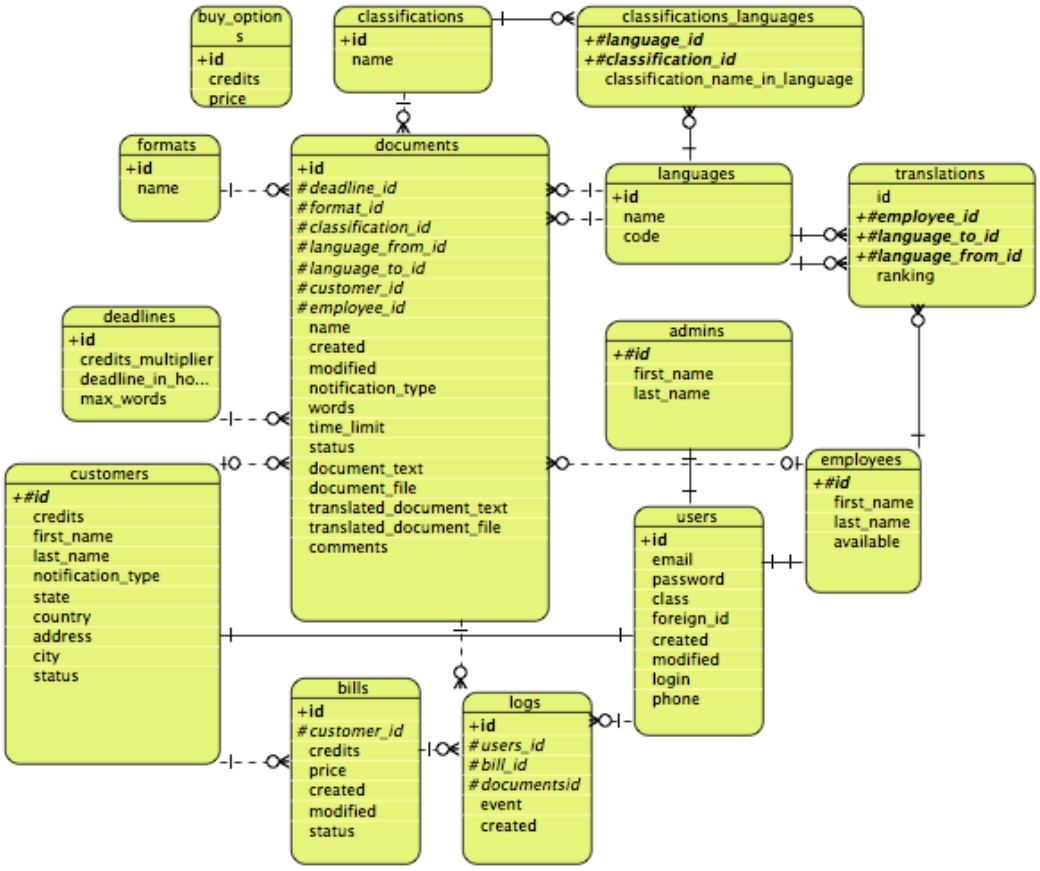


Figure 10.2: Final ER Diagram

10.1.2 The Web Application

Because the web application in essence is a typical web page built with PHP/HTML, it works in a normal client/server fashion. The application resides on a web server which accepts requests from users and responds depending on the request. The web server processes the request and uses the application to create a response to the user.

The application itself consists of different modules. When the web server gets the request, it is passed on to the CakePHP Dispatcher. The Dispatcher evaluates the

request and passes it on to the correct controller. The controller then gets information from different models, and passes it on to the view, which in turn is sent back to the person who sent the request. The structure described can be seen in Figure 3.2.

The main goal of the web application is to let the customers upload document for proofreading or translation and then get it back within a given time limit. For this to work there has to be some employees that can download the document uploaded by the customer, translate or proofread it and then upload it back to the server. Figure 6.3 shows the data flow between the two actors.

As stated before there are three different users for the web application: The customer, the employee and the administrator. Each of these have different functionality and the most important ones are:

Customer:

- Upload plain text, doc files, docx files, pdf files or odt files for proofreading or translation. See figure 7.8.
- Write in text in a text box and upload it for proofreading or translation.
- Buy more credits. The customer needs this to pay for the proofreading and translation. See figure 8.4 and figure 7.4.
- Download finished translated or proofread documents. See figure 6.7.

Employee:

- Set their status to away or available. This affects the possible languages the customer can get a document proofread or translated in. See figure 8.8.
- Download tasks given to them to start proofreading or translating it.
- Remove tasks from their task list. This is for when an employee suddenly can not work for some reason. Other employees will have to take this task manually later.
- Take tasks. Every employee have a list on their account of tasks that suits their skill and are not taken from any other employee. See figure 7.7.
- Upload documents that have been proofread or translated. The customer will now have this document available for download at their page.

Administrator:

- Can add and delete administrators to the system. See figure 8.9.
- Can add and delete employees to the system. See figure 8.9
- Can add and delete skills to the employees. See figure 8.7.
- Can see every customer and document on the server.

- Can delete customers. See figure 7.3.
- Can edit employee and customer details. See figure 7.2.

Figure 10.3 is a screenshot of the home page of the web application. You can find the rest of the screenshots in the appendix E.1.

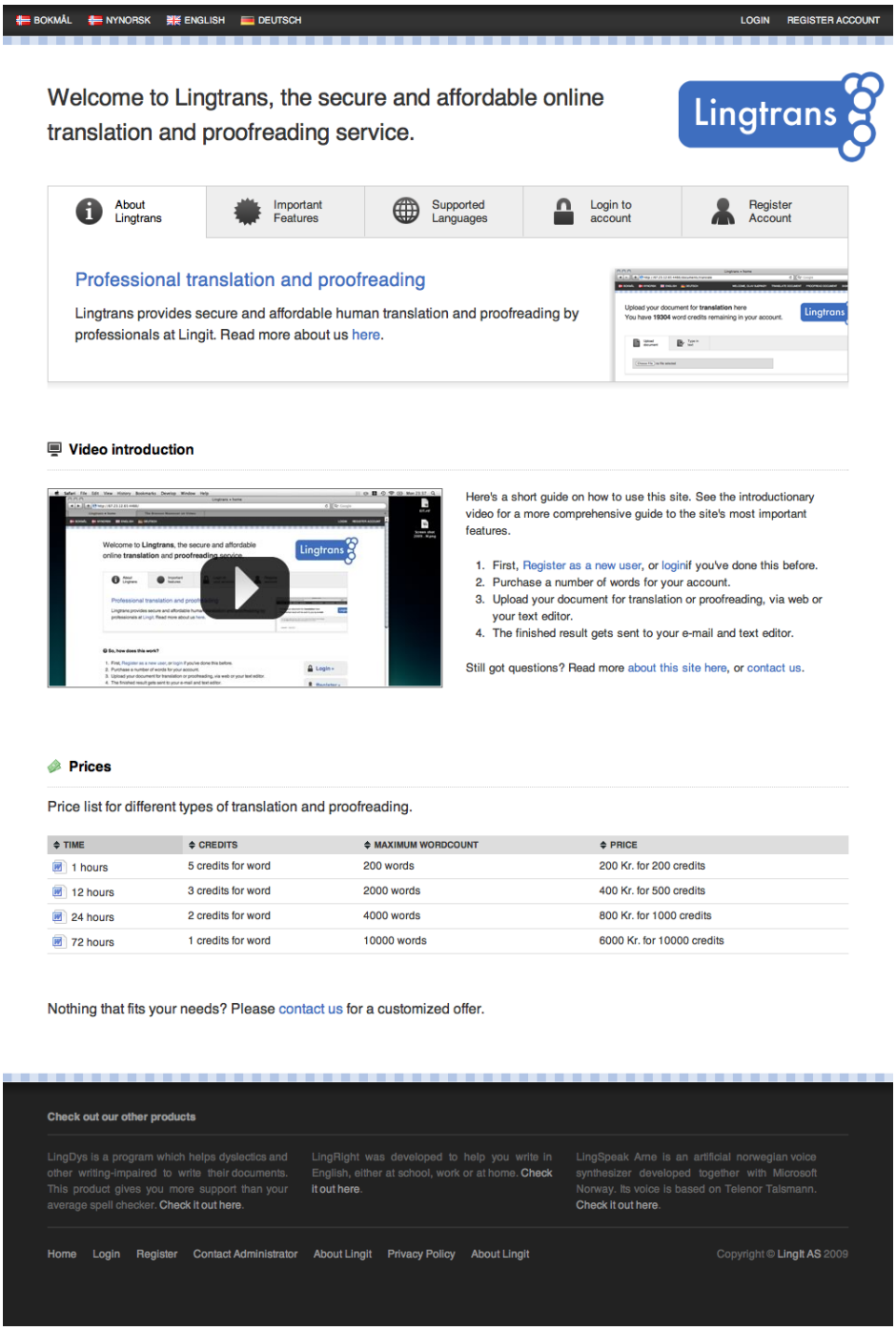

BOKMÅL
NYNORSK
ENGLISH
DEUTSCH
LOGIN
REGISTER ACCOUNT
Welcome to Lingtrans, the secure and affordable online translation and proofreading service.
Lingtrans
About Lingtrans
Important Features
Supported Languages
Login to account
Register Account
Professional translation and proofreading
Lingtrans provides secure and affordable human translation and proofreading by professionals at Lingit. Read more about us here.
Video introduction
Here's a short guide on how to use this site. See the introductionary video for a more comprehensive guide to the site's most important features.
1. First, Register as a new user, or loginif you've done this before.
2. Purchase a number of words for your account.
3. Upload your document for translation or proofreading, via web or your text editor.
4. The finished result gets sent to your e-mail and text editor.
Still got questions? Read more about this site here, or contact us.
Prices
Price list for different types of translation and proofreading.
TIME
CREDITS
MAXIMUM WORDCOUNT
PRICE
1 hours
5 credits for word
200 words
200 Kr. for 200 credits
12 hours
3 credits for word
2000 words
400 Kr. for 500 credits
24 hours
2 credits for word
4000 words
800 Kr. for 1000 credits
72 hours
1 credits for word
10000 words
6000 Kr. for 10000 credits
Nothing that fits your needs? Please contact us for a customized offer.
Check out our other products
LingDys is a program which helps dyslectics and other writing-impaired to write their documents. This product gives you more support than your average spell checker. Check it out here.
LingRight was developed to help you write in English, either at school, work or at home. Check it out here.
LingSpeak Arne is an artificial norwegian voice synthesizer developed together with Microsoft Norway. Its voice is based on Telenor Talsmann. Check it out here.
Home
Login
Register
Contact Administrator
About Lingit
Privacy Policy
About Lingit
Copyright © Lingit AS 2009

Figure 10.3: Home Page

10.1.3 The Microsoft Word Add-in

The result for the final user in the Microsoft add-in for Office Word is the same as for the web application, only the platform is different. The user runs Word like normal. Everything that has to be sent is sent as XML to an API on the server, which only then interacts with the database. Then the Web Application goes on for the rest. Figure 10.5, 10.6, 10.7 and 10.8 shows screenshots of the add-in.

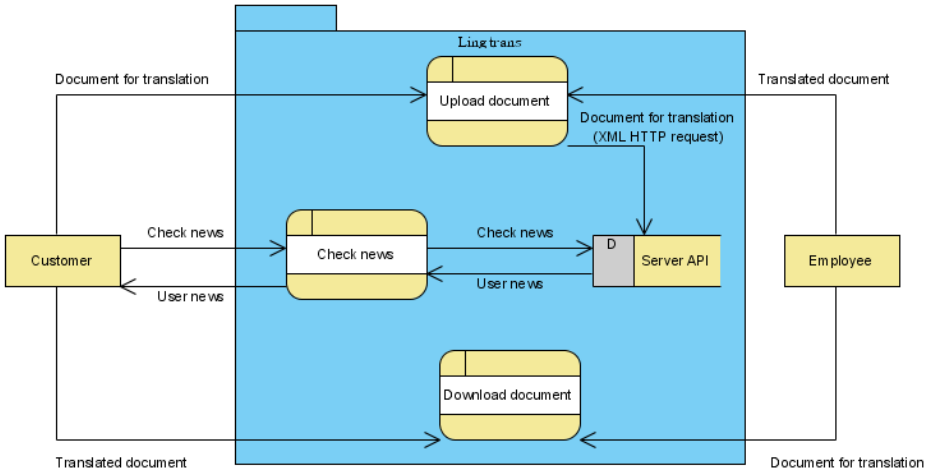


Figure 10.4: Data Flow Diagram for the Microsoft Office Word Add-in

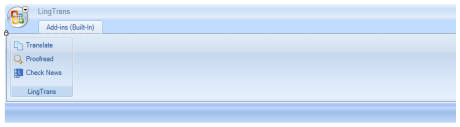


Figure 10.5: Office Ribbon Tab and Buttons for the Commands of the Add-in

10.2 Further Development

Even though the applications as of now can be seen as final versions, it may still be of interest to develop them further. A good application or service will always be under constant development to assure customer satisfaction and continued interest. We have therefore compiled a small list of possible future changes that may enhance the customer satisfaction or ease the work for employees or administrators.

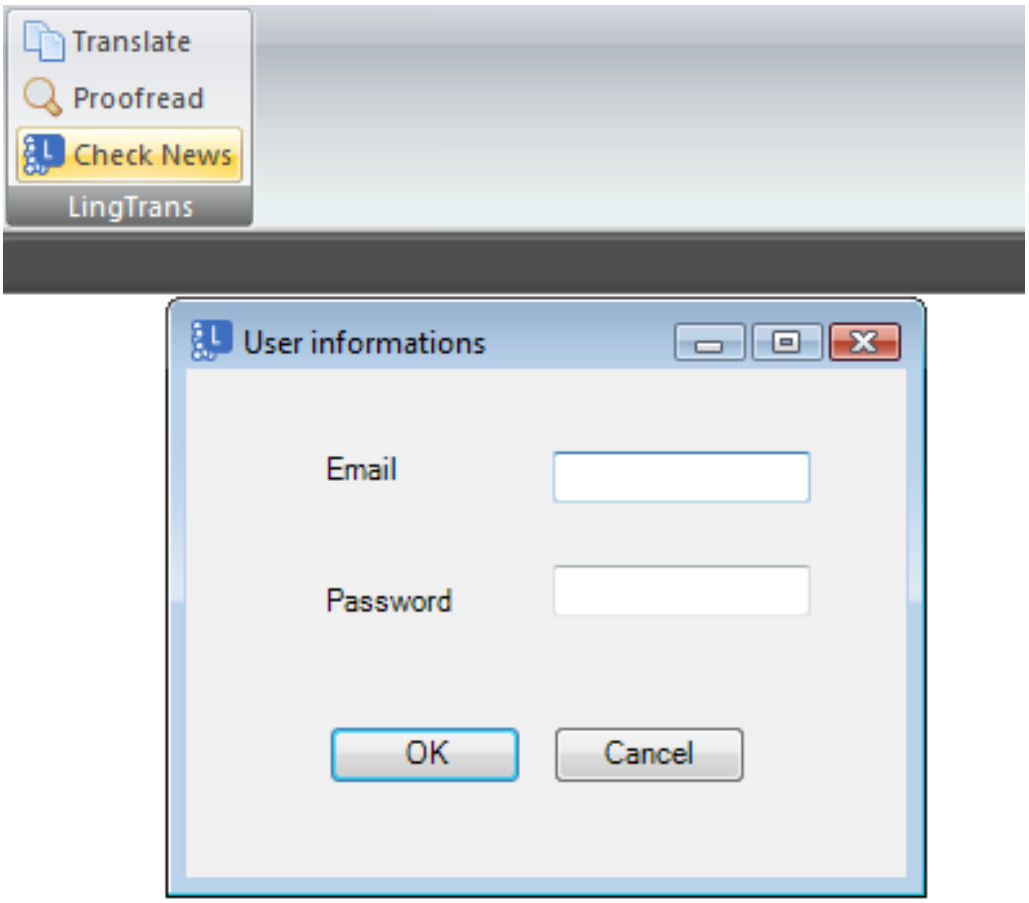


Figure 10.6: Check News Use Case Enabled

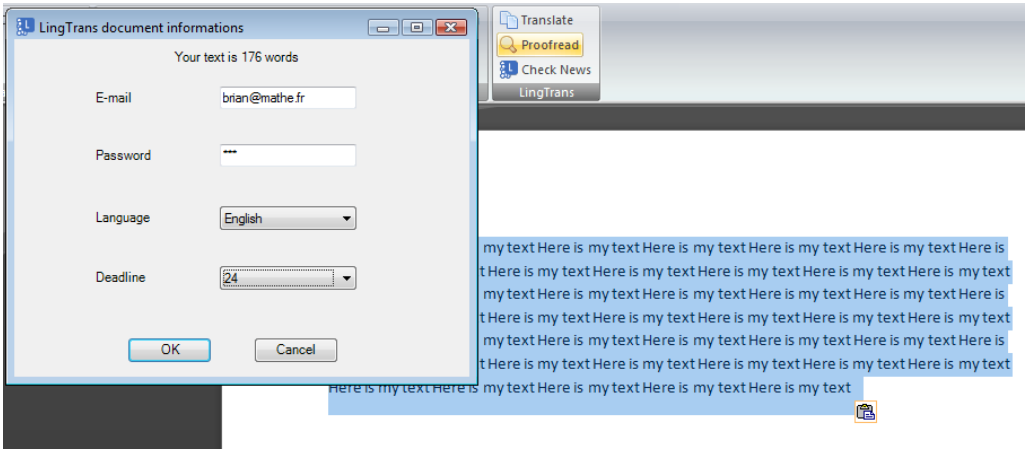

LingTrans document informations
Your text is 176 words
E-mail
brian@mathe.fr
Password
Language
English
Deadline
24
OK
Cancel
Translate
Proofread
Check News
LingTrans
my text Here is my text Here is my text Here is my text Here is my text Here is
t Here is my text Here is my text Here is my text Here is my text Here is my text
my text Here is my text Here is my text Here is my text Here is my text Here is
t Here is my text Here is my text Here is my text Here is my text Here is my text
my text Here is my text Here is my text Here is my text Here is my text Here is
t Here is my text Here is my text Here is my text Here is my text Here is my text
Here is my text Here is my text Here is my text Here is my text

Figure 10.7: Proofread Use Case Enabled

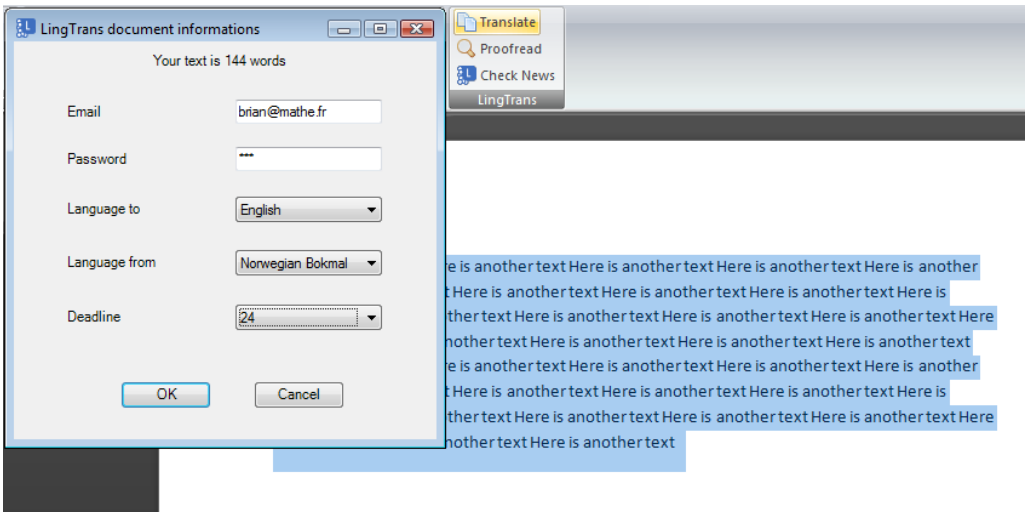


Figure 10.8: Translate Use Case Enabled

10.2.1 Web Application

This section contains the possible further development for the web application.

Scheduling

One way of extending the application is integrating some sort of time management system into it. This could serve as a way of making sure that there always will be some translators and proofreaders available for all languages. Right now it keeps track of this by warning administrators if there are no one available for a language, but by extending it one can prevent this from happening before it actually happens. There are many ways of implementing such a feature. It can be done automatically by the system by some sort of algorithm that ensures the availability of translators and proofreaders for all languages, or it may just the ability to insert such information into the database by an administrator. The first solution would be very complex and time consuming to implement, and can be done by a scheduling algorithm. The second solution can be implemented as easy as just creating fields where an administrator does the work of scheduling instead of an algorithm.

The Billing System

The billing system is as of now a temporary system that has the ability to classify bills as either unsent or sent. Invoices reside in a table in the database, but this is in no way a complete system. One can either implement support for existing invoice systems or extend the one currently in the application. The first alternative might make it possible to integrate the system they already use for their other applications with the one for this. The second alternative will make Lingtrans more independent, but will require a

lot of development to perfect.

In addition to this it would be possible to extend the payment options with Visa, MasterCard, PayPal or some similar system. This may ease the process of buying words, and may also ease the implementation of a billing system considering all the pre made systems out there.

Support For More File Types

The system currently supports files with the following types: DOC, DOCX, ODT and PDF. Even though these are the most common file types it may be desirable to support others as well for increased flexibility. An example is RTF (Rich Text format), which is another commonly used format by at least Windows and Mac OS users. Adding support for new file types requires functionality for decoding and extracting text from the file to be able to count the number of words.

Extension for OpenOffice Writer

Just like Microsoft Word, OpenOffice Writer has its own extension system. Sun has developed a set of bindings for different languages, but using a cross platform language is of course preferable, since this will include customers on platforms such as Linux and Mac OS as well as Windows.

User Confirmation

As of now, users can register and start buying words without confirming their true identity. Because the system needs to send invoices to customers, it can be desirable to have some sort of confirmation before they are allowed to use the system. This can be done in many ways, where one of the easiest and most widespread ways today is e-mail confirmation. This usually means that an e-mail is sent to the specified e-mail address when the account is created, and the customer will have to follow some sort of instructions to activate his account. This will not ensure validity of the registered address, but at least gives some sort of security.

Another important subject here is the use of CAPTCHA [23]. CAPTCHA is a system to ensure requests are generated by human beings and not some kind of robot or computer. This has been used with huge success in a range of internet applications, including forums, blogs and wikis [23]. Because CAPTCHA's are generally considered hard, or harder, to read, there might be a few problems implementing it in Lingtrans. Because this service is especially directed towards users with reading disabilities, implementing CAPTCHA might give customers trouble with registering. There are alternatives, such as audio CAPTCHA, which gives the user the ability to hear the word as speech as

well. This will aide people with reading disabilities, but only if they have speakers.

10.2.2 Microsoft Word Add-in

This section contains the further development for the Microsoft Word Add-in.

Portability to OpenOffice

At the end of the project, there has been no implemented add-in for OpenOffice, as we lacked some time to consider starting developing it. As a future implementation of the system, it would be a good idea to have this type of add-in for OpenOffice.

Directly download the results of the employees into the Word document

When a user is working with the add-in version of the system, it would be very comfortable for him/her to implement a method on the API that sends the document to the user automatically when the job with this text is over, to then incorporate it in the document he is working on at the moment. Of course, the user is not working on the document all the time so such a function would not be useful all the time but still relevant when possible.

Right-click function

For now, the different possibilities offered by the Word add-in are only possible with the button on the ribbon in the add-ins tab. It would be pretty comfortable for the final user to be able to select the functions on a right-click menu event, just after having selected his/her text.

Remember the texts when not connected to the internet

If a user of the add-in is not connected to the internet but still wants to tell the system about some documents he/she wants to be translated or proofread, it would be interesting to create a function that remembers the portion of text he/she wants to submit and send it the next time the user is connected to the internet.

10.3 Limitations

This section will try to enlighten some of the current limitations of the Web Application and the Microsoft Word add-in.

10.3.1 System Limitations

Normally when writing applications one end up with system limitations. For example, Java applications can not be run without Java Runtime, which can be seen as a limitation for the user. The same goes for C# applications, which needs the .net framework to function [22]. Our main application, the web application, does not need any special software installed on the customers computer to work, with the exception of a modern web browser. In this case, modern means that it has to understand HTML and CSS and have the ability to display images. Table 4.5, item 2.4, explains which browser engines the web site is optimized for. The tested browsers covers at least 99,41% according to a huge survey [15], and will in theory work with all others as well as long as they follow standards put down by The World Wide Web Consortium (W3C).

10.3.2 Supported Formats

Because of the nature of our system, new file formats can not be added without further developing the application. This is because the word count of the document is checked against the number of credits the customer currently has, which means that we have to be able to extract the text from the document. This may be easy for some formats, but harder for others. Reading pure text files is fairly straight forward, while for example text exported as images would require implementing some sort of optical character recognition, which is both difficult and time consuming. This also gives some requirements to the server hardware wise. If lots of customers are uploading documents at the same time, PHP will need a lot of memory to hold all the documents while counting words.

10.3.3 Javascript

The Web Application uses Javascript some places to make the it feel more interactive. While it is possible to use the web page without Javascript enabled, it will definitely work a lot better with. According to W3Schools [18] about 95% had Javascript enabled in 2008. This is a bit misleading though, because some users use browsers without support for Javascript, and those who turn Javascript off themselves are often more advanced users. Some of these users are using mobile phones, PDAs or other devices without Javascript support to browse the internet, which is not part of our audience.

CHAPTER 11

Project Evaluation

The project started with a very vague idea of how the final product would look like. Lingit had a general idea, but feedback and ideas from us was just as important for them as the ideas they had. It has therefore not been a project with a specific task, but more of a process with emphasis on brainstorming and development of a business idea.

The overall goal was to end up with a working application, but a lot of the time was spent finding out what it actually would look like in the end.

11.1 Using Scrum

Because of the nature of the task we quickly decided to use Scrum. Using Scrum would give us the opportunity to evaluate functionality alongside the project. It would be very hard to start collecting requirements on an application we only had a vague idea about how would look like in the end.

We therefore started by creating wireframes and continued by implementing functionality shown in the wireframes. The wireframes played a major role in helping us visualize the product and functionality, even though it was just a static representation. The application therefore became more dynamic and mature as the project evolved.

11.2 Group Dynamics

One of our first tasks was defining the different roles like group leader, coders, writers and so on. This was dealt with in plenum with every group member present, so that everyone would obtain a task they felt suited them. We started out following these roles in a quite strict manner, delegating tasks to the correct team members.

After two or three weeks it turned out that this was not working very well after all. Some members proved to be better coders than others, and some proved to better writers. The roles sort of automatically changed to a more natural partition without anyone of us really noticing it. When we started out we tried to divide work on the web application for almost all group members. What this caused was one of the team member, who had the most experience in web development, was running around and helping everyone else with their problems. The effect of this was that he could not do so much work himself and therefore the potential we had in him was lost. As this evolved the most experienced group member did all the php coding himself. This turned out to be much more effective. Of course he did not do everything himself, but the most of the difficult tasks. The rest of the group where mostly making sequence diagrams and documenting his work. While one member only worked on the Microsoft Word Add-on. We became aware of it later on in the project during an evaluation of how our work had evolved during the project, and continued with it as it seemed to make us a lot more productive as a group.

We decided early on to mostly work together instead of splitting the work and working from home. This was due to a number of reasons, but most importantly to keep a high level of motivation to and ensure that no one would fall behind. It turned out to be a great decision that was kept more or less throughout the whole project.

Another challenge regarding the group dynamics was our individual language skills. With four Norwegian team members, one from France and one from Italy, it was clear that English would be our language of choice, both for documents and verbal communication. Needless to say, this was a bit difficult in the beginning, leading to hardships in explaining different problems and situations. However, after a few weeks, this became routine as our communication and writing skills in English notably increased.

11.3 Learning PHP and CakePHP

Lingit required that we used PHP for the web application. Three of the six group members had used this language before, though not in any extensive projects. Also, as none of us had ever used a PHP framework, learning CakePHP would be an additional challenge.

As most of the group members have extensive experience in programming, learning PHP itself went pretty well. Learning the CakePHP framework however, was a bit more challenging. It is essential to understand its use of MVC and built in functions to use it properly. This meant that code we wrote when starting out using Cake often had to be rewritten as we learned new features of the framework. As a result of this, if we had started coding the system from scratch today, many of the features would be coded and organized differently.

11.4 Risks

The only risk that really made an impact on the project was illness. The fall is typically the time of the year most people get ill, and with the arrival of the widely known swine flu it proved to be a bigger problem than normal. All of the group members was ill at least once during the project. This was solved with the use of instant messaging, which enabled us to keep in touch throughout the whole day even though we were at physically different locations. The sickdays still set us back a bit, but the use of instant messaging helped a lot with limiting the extent.

Another problem we encountered during the project was inexperience with the chosen programming language. This, though, never really made an impact on the project because of the large amount of documentation available for each language on the Internet.

11.5 Time Estimation

Our initial time estimation was a total of 1872 hours. This was calculated by simply adding up all the hours for each person for the whole project, which was set to 24 hours each week. Initially this seemed like an extreme amount of hours to spend on a project, but it would later prove not to be that extreme after all.

Including the work done on the report and the implementation of our two applications, we ended up spending a total of 1727 hours in total. Split among the six of us, this is rounded down to 22 hours per week. So we have used less time than the initial estimation. This is because we have had very few problems during the project. 1872 does not feel so extreme, but we still feel that it is a lot of time to use on this project and we think that we still could have used less time than we did since we have for some periods been a bit inefficient.

Figure 11.1 shows how the total work has been divided among different tasks. As you can see from the diagram we have spent most our time implementing or writing the report.

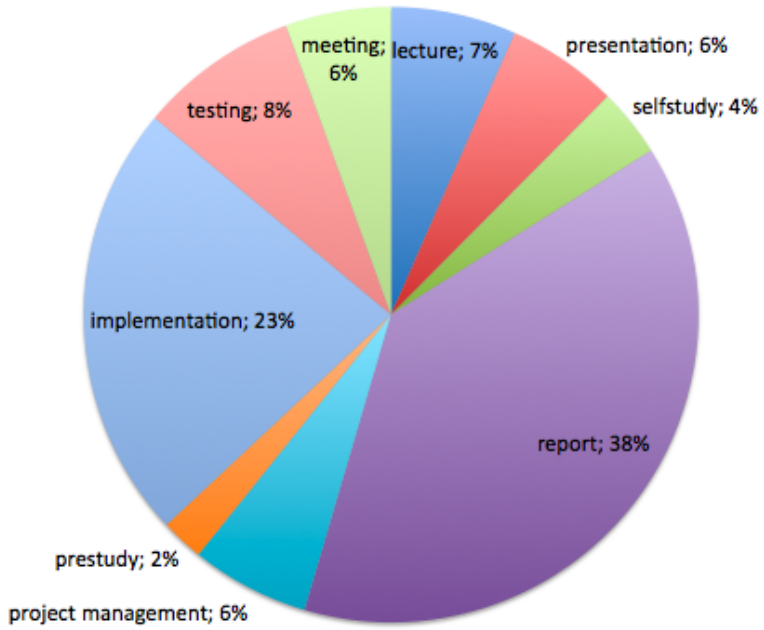


Figure 11.1: Cake Diagram of Hours Spent

Figure 11.2 shows how many hours the group have put into the project every week. The diagram is a bit misleading as the first week show very little work. The reason for this is that we did not start the project before tuesday and we had one team member less the first week. From the diagram you can also see that we started out with a lot of work in the beginning. This has it's natural causes. In the start we really wanted to get a better overview of the project and therefore we used a lot of hours on pre-study and just getting structured in the group.

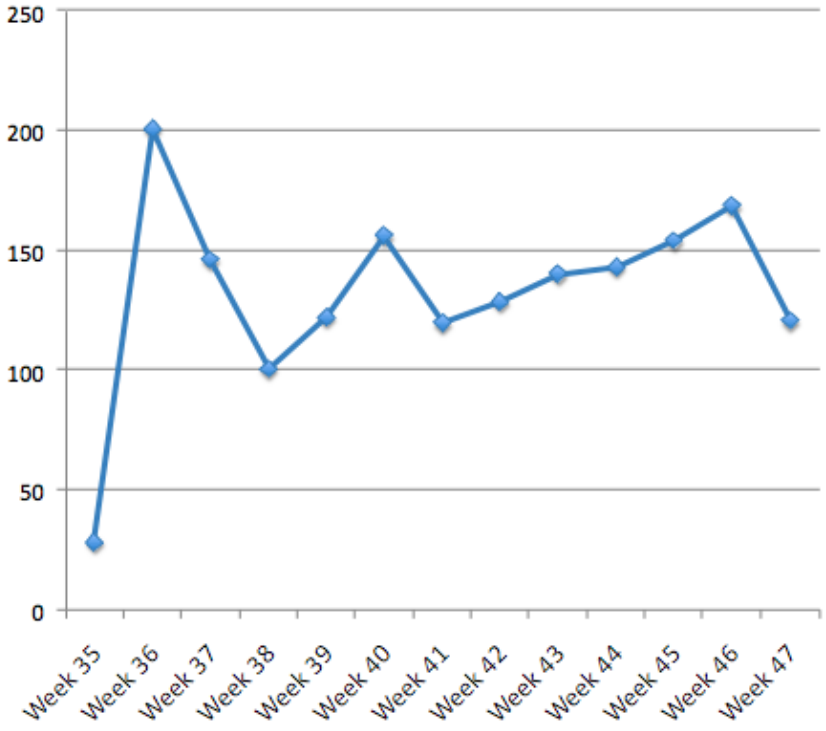


Figure 11.2: Column Diagram of Hours Spent Each Week

Part IV

Definitions, References & Appendices

CHAPTER 12

Acronyms

- **NTNU:** The Norwegian University of Science and Technology.
  Located in Trondheim. Being the second largest of the seven universities in Norway, it has the main national responsibility for higher education in technology.
- **PHP:** PHP: Hypertext Preprocessor (Recursive Acronym).
  Widely used scripting language for web development
- **MySQL:** My Structured Query Language.
  Popular language to construct and query databases for web applications.
- **BusTUC:** Bus The Understanding Computer.
  BusTUC is a natural language problem solver capable of answering questions about bus departures in Trondheim stated in common english and norwegian.
- **SMS:** Short Message Service.
  A form of text messaging on mobile phones
- **API:** Application Programming Interface.
  Abstract document which specifies an interface and the behavior of the identifiers specified in that interface.
- **ER-diagram:** Entity-Relationship Diagram.
  A high-level conceptual data model, much used in designing databases.
- **SVN:** Subversion (version control system).
  It allows users to keep track of changes made to any type of electronic data, typically source code, web pages or design documents.
- **MVC:** Model View Controller.
  A architectural pattern in computer software development
- **HTML:** HyperText Markup Language.
  The predominant markup language for web pages. It provides a means to describe the structure of text-based information in a document

- **CMS:** Content Management System.
  A system for managing content and providing it in various formats, used mainly for websites
- **BSD:** Berkeley Software Distribution.
  Is the UNIX derivative distributed by the University of California, Berkeley, starting in the 1970s.
- **MIT:** Massachusetts Institute of Technology.
  The MIT License is a free software license originating at the Massachusetts Institute of Technology (MIT). Specifically, it is a GPL-compatible permissive license, meaning that it permits reuse within proprietary software on the condition that the license is distributed with that software.
- **ORM:** Object-relational mapping.
  A programming technique for converting data between incompatible type systems in databases and object-oriented programming languages.
- **URL:** Uniform Resource Locator.
  Uniform syntax for global identifiers of network-retrievable documents. Example: http://www.lingit.no
- **AJAX:** Asynchronous Javascript And XML.
  A web development technique used for creating interactive web applications. It strives to communicate as little data with the server as possible, to avoid reloading of entire web pages for every user request. This is intended to increase the web page's interactivity, speed, functionality, and usability.
- **PDF:** Portable Document Format.
  File format created by Adobe Systems in 1993 for document exchange.
- **VBA:** Visual Basic for Applications.
  is an implementation of Microsoft's Visual Basic, an event driven programming language and associated integrated development environment (IDE) which is built into most Microsoft Office applications.
- **C#:** C sharp.
  An object-oriented programming language developed by Microsoft as part of the .NET initiative.
- **OO:** Object Oriented.
  A programming paradigm that uses "objects" and their interactions to design applications and computer programs.
- **IE6:** Internet Explorer 6.
  Graphical web browsers developed by Microsoft and included as part of the Microsoft Windows line of operating systems starting in 1995.
- **XML-RPC:** XML Remote Procedure Call.

A remote procedure call protocol which uses XML to encode its calls and HTTP as a transport mechanism.

- **XML:** Extensible Markup Language.
  A general-purpose markup language. It is classified as an extensible language because it allows its users to define their own tags.
- **COM:** Component object model.
  A platform for software componentry introduced by Microsoft in 1993.
- **VSTO:** Visual Studio Tools for Office.
  A set of development tools that allows extensions to the Office applications to be written in .NET languages.
- **.NET:** Microsoft XML Web Services platform.
  A software framework that can be installed on computers running Microsoft Windows operating systems. The .NET Framework is a Microsoft offering and is intended to be used by most new applications created for the Windows platform.
- **ISO:** International Organization for Standardization.
  An non-governmental international-standard-setting body composed of representatives from various national standards organizations.
- **NB/NN:** Norwegian Bokmål / Norwegian Nynorsk.
  The ISO-standard short form of the norwegian languages bokmål and nynorsk.
- **CAPTCHA:** Completely Automated Public Turing test to tell Computers and Humans Apart.
  Widely used in web applications to ensure that the response is not generated by a computer, for example registration and ordering of products through the web.
- **CSS:** Cascading Style Sheets.
  a style sheet language used to describe the presentation semantics (that is, the look and formatting) of a document written in a markup language (see HTML).
- **W3C:** The World Wide Web Consortium.
  The main international standards organization for the World Wide Web.
- **BLOB:** Binary Large Objects.
  A collection of binary data stored as a single entity in a database management system. Blobs are typically images, audio or other multimedia objects.

Definitions are taken from [2] and [17].

CHAPTER 13

Glossary

- **Microsoft Word Macro:** this is the prototype add-on made in VBA.
- **Microsoft Word add-in:** this the add-in to Microsoft Word made in C#.
- **Sprint:** this is the term for one iteration of our implementation. This is a term from the Scrum agile development method. Every sprint should end with a working product.
- **Customer:** the customer user of the web application.
- **Employee:** the employee user of the web application. A translator, proofreader, or both, working for Lingit.
- **Administrator:** the administrator user of the web application. Person working for Lingit with full application rights: register new employees, administrators and delete users are some of the rights.
- **Client:** refers to Lingit, our 'customer' during this project.
- **Instant Messaging:** real-time communication between two or more persons exchanging text messages through the Internet.
- **Git:** the version control system used in this project. (See SVN in and MIT Licence acronyms 12).
- **LateX:** the software tool used to write this report.
- **Prototype:** one of the first units, and usually a working model, of a new product.
- **Scrum:** an iterative incremental method for agile software developemnt and work management.
- **Product backlog:** part of the Scrum agile development process. High-level document descriping the overall to do list of a project.
- **Sprint backlog:** part of the Scrum agile development process. Document describing work to be done in an upcomming sprint.

- **Burn down chart:** sprint backlog items are estimated in working hours and the remaining work to be done in the sprint is calculated and visualized in a burn down chart.
- **Gantt chart:** illustrates the project schedule, including milestones and strat and finish dates
- **Plugin:** an auxillerary software application that extends functionality or capabilities of an existing sotware application.
- **CakePHP:** web application framework written in PHP. Makes it simpler to write robust web applications. (See PHP and MVC in acronyms 12).
- **Apache:** very common web server. Serve both static content and dynamic Web pages on the World Wide Web (See PHP and MySQL in acronyms 12).
- **Linux:** free and open source operation system.
- **PHP Framework:** see CakePHP.
- **OpenOffice:** free and open source text processing software.
- **Use case:** techique in software engineering to describe system behaviour and functionality. What a user of the system is able to do with it.
- **Server:** computer program that deliver a service to clients.
- **Wireframe:** visualization of the interface design without any implemented features. Provides suggestion for the structure of a website and the relationship between the pages it contains.
- **Data flow diagram:** graphical representation of the flow of data in an information system.
- **Class diagram:** graphical representation of the relationship, structure and attributes, between entities in a system.
- **Sequence diagram:** graphical representation of how processes operate with one another.
- **Localization:** the process of making a website country-specific with regards to the spoken language of that contry.
- **Eclipse:** a software development enviroment which is primarily used to develop applications written in Java.

References

[1] Mark S. Dorfman. *Introduction to Risk Management and Insurance*. N.J: Prentice Hall, 9th edition edition, 2007.

[2] en.wikipedia.org. Wikipedia - the free encyclopedia. `http://en.wikipedia.org/wiki/Main_Page`, 2001.

[3] Eric Lippert Eric Carter. *Visual Studio Tools for Office 2007: VSTO for Excel, Word, and Outlook*. Addison-Wesley Professional, 2009.

[4] IBM. The future of php. `http://www.ibm.com/developerworks/opensource/library/os-php-future/`, 2008-05-06.

[5] ieee.org. Ieee standard for software test documentation. `http://ieeexplore.ieee.org/xpl/tocresult.jsp?isNumber=16010`, 1998-09-16.

[6] Cake Software Foundation Inc. Cakephp. `http://cakephp.org/`, 2009.

[7] Cake Software Foundation Inc. Cakephp conventions. `http://book.cakephp.org/view/22/CakePHP-Conventions`, 2009-09-23.

[8] Cake Software Foundation Inc. Cakephp mvc request. `http://book.cakephp.org/view/10/Understanding-Model-View-Controller`, 2009-09-23.

[9] Jon Espen Ingvaldsen Steinar Hagen Jon Atle Gulla, Reidar Conradi and Geir Solskinnsbakk. Introduction to course tdt4290 customer driven project. `http://www.idi.ntnu.no/emner/tdt4290/docs/2009/TDT4290InfoBook2009.pdf`, 2009-21-10.

[10] Jakob Nielsen. The need for speed. `http://www.useit.com/alertbox/9703a.html`, 2009-10-13.

[11] Jakob Nielsen. Scrolling and scrollbars. `http://www.w3schools.com/browsers/browsers_stats.asp`, 2009-10-13.

[12] php.net. Php 5 changelog. `http://php.net/ChangeLog-5.php`, 2009-11-02.

[13] Scott Schnoll. Safety and security on the internet. `http://www.nwnetworks.com/sands.htm`, 2009-10-13.

[14] scrummethodology.com. Scrum methodology. `http://scrummethodology.com/`, 2009.

[15] Market Share. Browser market share. http://marketshare.hitslink.com/browser-market-share.aspx?qprid=0, 2009-11-02.

[16] Synfony. Open-source php web framework. http://www.symfony-project.org/, 2009.

[17] thefreedictionary.com. The free dictionary. http://www.thefreedictionary.com/, 2009.

[18] W3Schools. Browser statistics. http://www.w3schools.com/browsers/browsers_stats.asp, 2009-10-13.

[19] W3Schools. Screen resolution statistics. http://www.w3schools.com/browsers/browsers_display.asp, 2009-10-13.

[20] Wikipedia. Plug-in (computing). http://en.wikipedia.org/wiki/Plug-in_(computing), 2009.

[21] Wikipedia. Php. http://en.wikipedia.org/wiki/PHP, 2009-11-01.

[22] Wikipedia. C sharp (programming language). http://en.wikipedia.org/wiki/C_Sharp_(programming_language), 2009-11-02.

[23] Wikipedia. Captcha. http://en.wikipedia.org/wiki/CAPTCHA, 2009-11-02.

[24] Wikipedia. Content mangement system. http://en.wikipedia.org/wiki/Content_management_system, 2009-11-02.

[25] Zend.com. About zend framework. http://framework.zend.com/about/overview, 2009.

APPENDIX A

Project Directive Appendix

A.1 Templates

Here are the templates for client meeting agendas and minutes. The same for the supervisor meetings are also included.

Client Templates

Notice of customer meeting
Date:

Lingit AS

Information
Date: xx.xx.2009
Time: xx:xx
Place: Teknobyen

Agenda

- case 1.
- case ...
- case n.

Minutes from client Meeting

Date: xx.xx.2009 Time: xx:xx
Place: Room X

Attendees

Present: Bendik Solheim, Brian Mathé, Erik Andreas Larsen, Kjetil Aamodt, Olav Frihagen, Bjørkøy, Riccardo Ortolan. **Client:** Tor Rune Skoglund. **Not present:** None

Minutes

- case 1.
- case ...
- case n.

To-do list

- case 1.
- case ...
- case n.

Supervisor Templates

Notice of customer meeting
Date:

Lingit AS

Information
Date: xx.xx.2009 Time: xx:xx
Place: Room X

Agenda

- case 1.
- case ...
- case n.

Minutes from Meeting

Date: xx.xx.2009 Time: xx:xx
Place: Room X

Attendees

Present: Bendik Solheim, Brian Mathé, Erik Andreas Larsen, Kjetil Aamodt, Olav Frihagen, Bjørkøy, Riccardo Ortolan. **Supervisor:** Basit A. Kahn, Jahn M. W. Kristiansen. **Not present:** None

Minutes

- case 1.
- case ...
- case n.

To-do list

- case 1.
- case ...
- case n.

A.2 Test Case Descriptions

Table A.1: Document Controller Function Test

Title	Document Controller Function Test
Test Identifier	UNTE-00
Dependencies	SYWE-00
Responsible	Erik Andreas Larsen.
Feature to be tested	Test if a customer and employee is able to upload and download documents.
Features included	1.3, 1.4, 1.7, 1.10, 1.11, 2.1
Reference(s)	UC-05, UC-06, UC-07, UC-11, UC-14, UC-16
Pre-conditions	Document controller and view must be implemented
Execution steps	Follow the scenario of the referenced use cases. 1. Upload empty documents. 2. Upload documents with unsupported extensions. 3. Upload correct documents.
Expected result	Only the correct documents should be uploaded. Else, the user should be prompted with an error message.

Table A.2: Email Notification Function Test

Title	Email Notification Function Test
Test Identifier	UNTE-01
Dependencies	UNTE-00
Responsible	Erik Andreas Larsen.
Feature to be tested	Test if an employee gets an email notification when there is new work available, and that a customer gets email notification when work is completed.
Features included	3.3, 2.12, 2.19
Reference(s)	UC-10, UC-17
Pre-conditions	UNTE-00 is sucsessfully tested.
Execution steps	Follow the scenario of the referenced use cases.
Expected result	Email is sent to the correct receivers.

Table A.3: Framework Controllers Tests

Title	Framework Controllers Tests
Test Identifier	UNTE-02
Dependencies	
Responsible	Riccardo Ortolan.
Feature to be tested	All the Framework controllers have been tested.
Features included	All the functions in the controllers.
Reference(s)	
Pre-conditions	Controllers must be implemented.
Execution steps	
Expected result	The tests should pass without returning any errors.

Table A.4: Framework Models Tests

Title	Framework Models Tests
Test Identifier	UNTE-03
Dependencies	
Responsible	Riccardo Ortolan.
Feature to be tested	All the Framework models have been tested.
Features included	All the functions in the models.
Reference(s)	
Pre-conditions	Models must be implemented.
Execution steps	
Expected result	The tests should pass without returning any errors.

Table A.5: Framework Helpers Tests

Title	Framework Helpers Tests
Test Identifier	UNTE-04
Dependencies	
Responsible	Riccardo Ortolan.
Feature to be tested	All the Framework helpers have been tested.
Features included	All the functions in the helpers.
Reference(s)	
Pre-conditions	Helpers must be implemented.
Execution steps	
Expected result	The tests should pass without returning any errors.

Table A.6: Framework Database Tests

Title	Framework Database Tests
Test Identifier	UNTE-05
Dependencies	
Responsible	Riccardo Ortolan.
Feature to be tested	All the Framework helpers have been tested.
Features included	All the functions in the Datasources, Schema and DbAcl.
Reference(s)	
Pre-conditions	Datasources, Schema and DbAcl must be implemented.
Execution steps	
Expected result	The tests should pass without returning any errors.

Table A.7: User Registration and Login Test

Title	User Registration and Login
Test Identifier	SYWE-00
Dependencies	
Responsible	Erik Andreas Larsen.
Feature to be tested	Test if a user is able to register on the website with an email, password, full name and invoice details.
Features included	1.1, 1.2, 1.8, 2.10
Reference(s)	UC-00, UC-01, UC-13, UC-18
Pre-conditions	User(Customer, Employee and Administrator) controller and view must be implemented
Execution steps	Follow the scenario of use case UC-00, UC-01. 1. Enter uncorrect input. 2. Enter correct input. 3. Login to system with the newly created account.
Expected result	The user input is successfully stored in the database only when correct input is entered. The user should be able to login to the system after registration.

Table A.8: Customer Account Settings Test

Title	Customer Account Settings
Test Identifier	SYWE-01
Dependencies	SYWE-00
Responsible	Erik Andreas Larsen.
Feature to be tested	Test if a customer is able to edit account setting on the website including email, password, full name and invoice details.
Features included	1.2,
Reference(s)	UC-04
Pre-conditions	User controller and view must be implemented
Execution steps	Follow the scenario of use case UC-04. 1. Enter uncorrect input. 2. Enter correct input.
Expected result	The user input is successfully stored in the database only when correct input is entered.

Table A.9: Administrator Management Test

Title	Administrator Management
Test Identifier	SYWE-02
Dependencies	SYWE-00
Responsible	Erik Andreas Larsen.
Feature to be tested	Test if an administrator is able to see, edit and delete customers, employees and other administrator trough the web interface.
Features included	1.2,
Reference(s)	UC-19, UC-20, UC-21, UC-22
Pre-conditions	Administrator controller and view must be implemented
Execution steps	Follow the scenario of the referenced use cases.
Expected result	All scenarios are successfully completed, and the results of the action is reflected in the database.

Table A.10: Employee Availability Test

Title	Employee Availability Test
Test Identifier	SYWE-03
Dependencies	SYWE-00
Responsible	Erik Andreas Larsen.
Feature to be tested	Test if an employee is able to set the availability status to away through the Web Application
Features included	2.7
Reference(s)	UC-13
Pre-conditions	Employee controller and default view must be implemented
Execution steps	Login as Employee(see use case UC-13). 1. Press set away button. 2. Press set available button.
Expected result	The employee is registered as away in the database, and this is displayed correctly by the default view.

Table A.11: Performance Test

Title	Performance Test
Test Identifier	PETE-00
Dependencies	All other system tests
Responsible	Erik Andreas Larsen.
Feature to be tested	Test the loading time for every web application page is within the maximum limit of 0,5 seconds
Features included	3.2
Reference(s)	
Pre-conditions	All page controllers and views must be implemented
Execution steps	Surf through all the webapplication pages and messure loading time, given by CakePHP debug level one.
Expected result	All pages load within 0,5 seconds.

Table A.12: Customer Usability Test

Title	Customer Usability Test
Test Identifier	USTE-00
Dependencies	
Responsible	Riccardo Ortolan.
Feature to be tested	Test if a customer is able to register, buy words and upload a document, in less than 10 minutes.
Features included	1.1, 1.2, 1.4, 1.5, 1.6,
Reference(s)	UC-00, UC-01, UC-03, UC-05, UC-09
Pre-conditions	User(Customer) controller and view must be implemented
Execution steps	Follow the scenario of use case UC-00, UC-01, UC-03. 1. Enter uncorrect input. 2. Enter correct input. 3. Login to system with the newly created account. 4. Buy new words. 5. Upload a new document to be translated.
Expected result	The user should be able to do these operations in less than 6 minutes.

Table A.13: Client Acceptance Test

Title	Client Acceptance Test
Test Identifier	ACTE-00
Dependencies	
Responsible	Riccardo Ortolan.
Feature to be tested	Test if a customer is able to register, buy words and upload a document, in less than 10 minutes.
Features included	1.1, 1.2, 1.4, 1.5, 1.6,
Reference(s)	UC-00, UC-01, UC-03, UC-05, UC-09
Pre-conditions	User(Customer) controller and view must be implemented
Execution steps	Follow the scenario of use case UC-00, UC-01, UC-03. 1. Enter uncorrect input. 2. Enter correct input. 3. Login to system with the newly created account. 4. Buy new words. 5. Upload a new document to be translated.
Expected result	The user should be able to do these operations in less than 10 minutes.

APPENDIX B

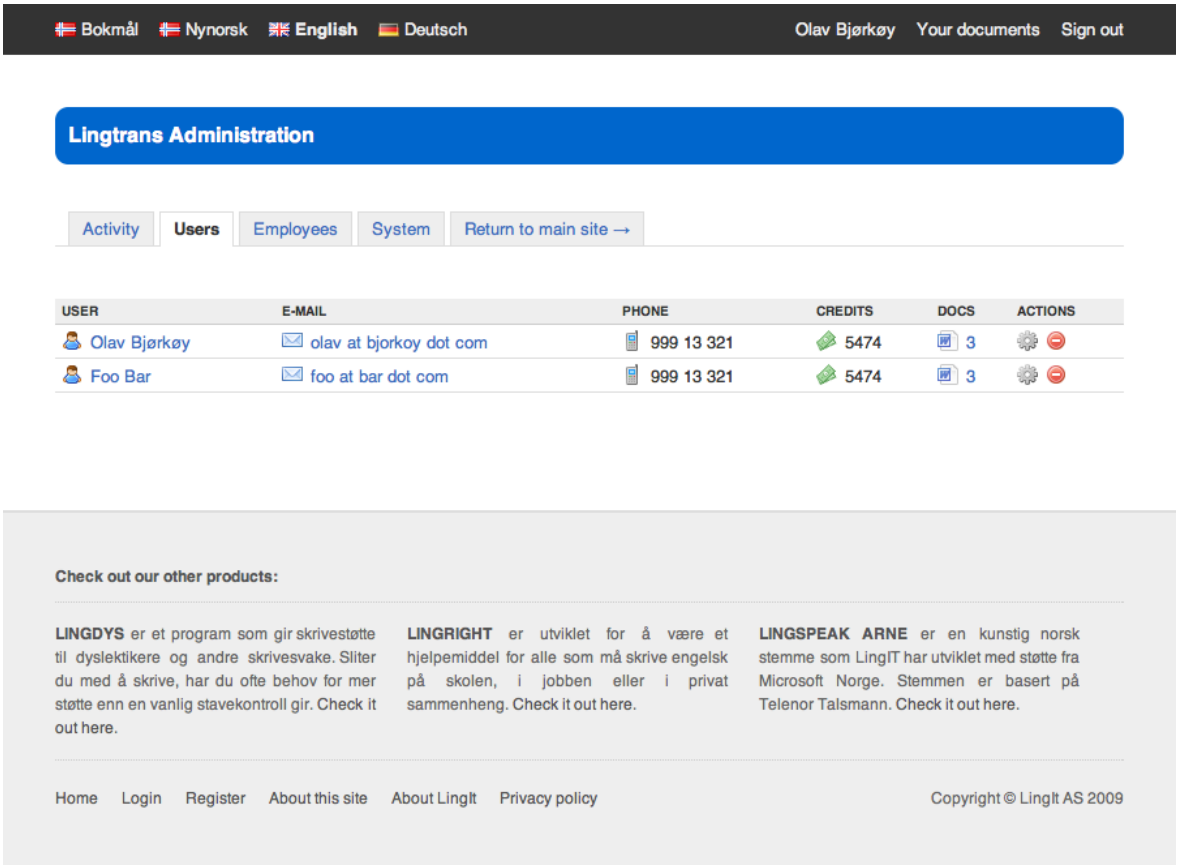
Sprint 1 Appendix

B.1 Wireframes

In addition to the wireframe shown in chapter 4.4, we have some other early screenshots as well. This is the very first version of the website, which was also shown to the customer.

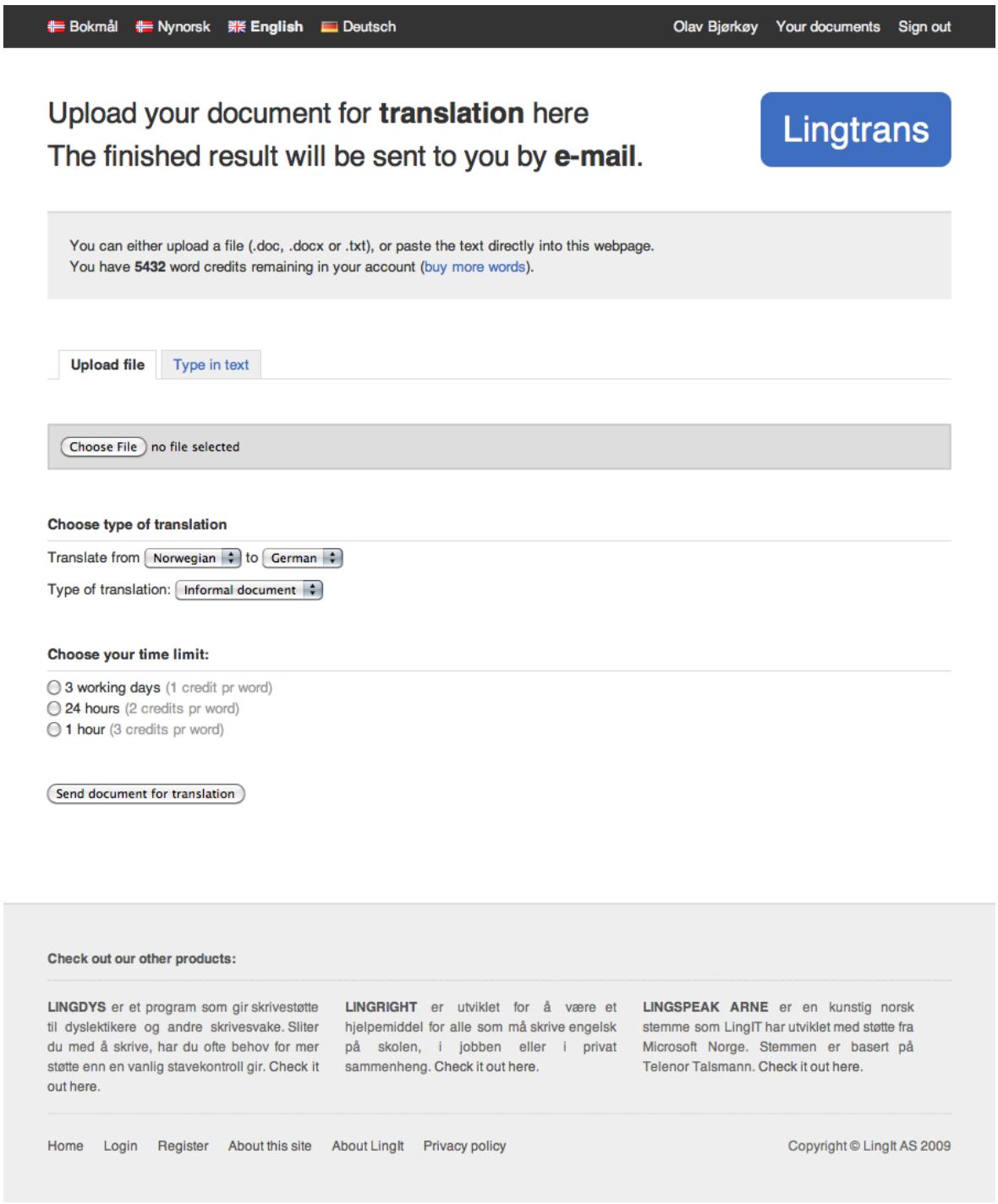
Admin page

This is the wireframes for the admin page. As you can see from the wireframe the admin can see the users and some information on them. He also have some other tabs he can select like: Activity, Employees and System. It is not yet clear what all these should include, but they give a general overview.



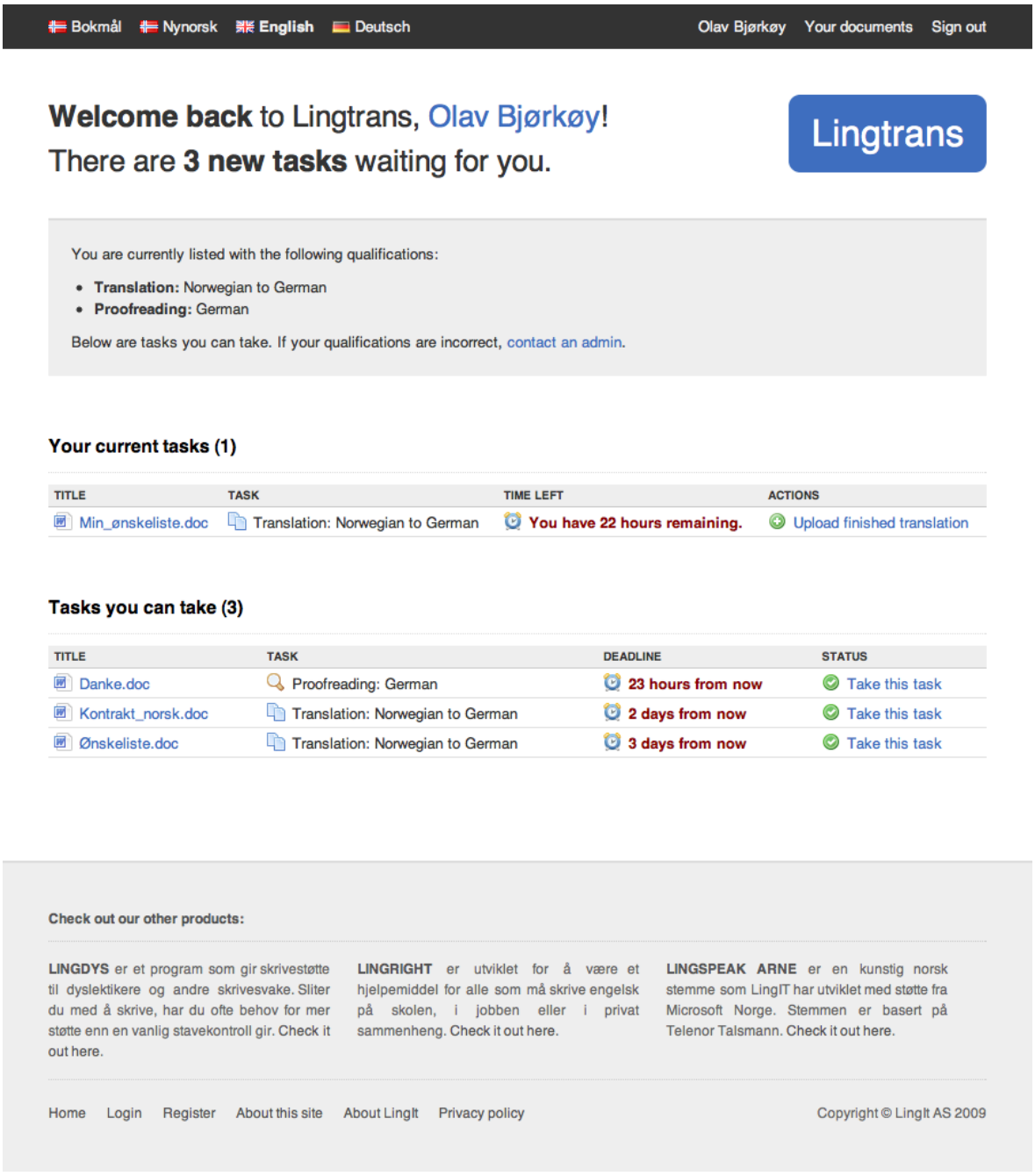
B.1.1 Translate page

This is what the user sees when he wants to upload a text for translation. The user can choose a file to upload or he can type the text in a text box. There are also some options for selecting translation language and response time for the translation.


Bokmål
Nynorsk
English
Deutsch
Olav Bjørkøy
Your documents
Sign out
Upload your document for translation here
The finished result will be sent to you by e-mail.
Lingtrans
You can either upload a file (.doc, .docx or .txt), or paste the text directly into this webpage.
You have 5432 word credits remaining in your account (buy more words).
Upload file
Type in text
Choose File
no file selected
Choose type of translation
Translate from Norwegian to German
Type of translation: Informal document
Choose your time limit:
3 working days (1 credit pr word)
24 hours (2 credits pr word)
1 hour (3 credits pr word)
Send document for translation
Check out our other products:
LINGDYS er et program som gir skrivestøtte til dyslektikere og andre skrivesvake. Sliter du med å skrive, har du ofte behov for mer støtte enn en vanlig stavekontroll gir. Check it out here.
LINGRIGHT er utviklet for å være et hjelpemiddel for alle som må skrive engelsk på skolen, i jobben eller i privat sammenheng. Check it out here.
LINGSPEAK ARNE er en kunstig norsk stemme som LingIT har utviklet med støtte fra Microsoft Norge. Stemmen er basert på Telenor Talsmann. Check it out here.
Home
Login
Register
About this site
About LingIt
Privacy policy
Copyright © LingIt AS 2009

B.1.2 Translator page

The translator will see this page when he logs in. The translator will see tasks he already are assigned and tasks he can take in the future.


Bokmål Nynorsk English Deutsch
Olav Bjørkøy Your documents Sign out
Welcome back to Lingtrans, Olav Bjørkøy!
There are 3 new tasks waiting for you.
Lingtrans
You are currently listed with the following qualifications:
Translation: Norwegian to German
Proofreading: German
Below are tasks you can take. If your qualifications are incorrect, contact an admin.
Your current tasks (1)
TITLE TASK TIME LEFT ACTIONS
Min_ønskeliste.doc Translation: Norwegian to German You have 22 hours remaining. Upload finished translation
Tasks you can take (3)
TITLE TASK DEADLINE STATUS
Danke.doc Proofreading: German 23 hours from now Take this task
Kontrakt_norsk.doc Translation: Norwegian to German 2 days from now Take this task
Ønskeliste.doc Translation: Norwegian to German 3 days from now Take this task
Check out our other products:
LINGDYS er et program som gir skrivestøtte til dyslektikere og andre skrivesvake. Sliter du med å skrive, har du ofte behov for mer støtte enn en vanlig stavekontroll gir. Check it out here.
LINGRIGHT er utviklet for å være et hjelpemiddel for alle som må skrive engelsk på skolen, i jobben eller i privat sammenheng. Check it out here.
LINGSPEAK ARNE er en kunstig norsk stemme som LingIT har utviklet med støtte fra Microsoft Norge. Stemmen er basert på Telenor Talsmann. Check it out here.
Home Login Register About this site About LingIt Privacy policy
Copyright © LingIt AS 2009

B.1.3 User page

This is the page the user will see when he is logged in. He can choose between translate, proofreading and purchase of new words. The user will also see all previous documents.

Bokmål Nynorsk English Deutsch Olav Bjørkøy Your documents Sign out

Welcome back to Lingtrans, Olav Bjørkøy!
You have **5432 words** remaining in your account.

Lingtrans

Lingtrans provides secure and affordable human translation and proofreading by professionals at LingIt.
Submit your documents for translation or proofreading below, or read more about us.

Things to do

- Translate a new document
- Proofread a new document
- Purchase more word credits

Your documents

You have 2 documents currently in progress:

TITLE	TASK	STATUS
Min_ønskeliste.doc	Translation from Norwegian to German	Currently being translated. At most 22 hours remaining.
Book_intro_draft.doc	Proofreading in German	Currently being proofread. At most 2 days remaining.

Here are your previous documents:

TITLE	TASK	STATUS
Anbud.doc	Proofreading in German	Finished proofreading. Download result here.
Kontrakt_norsk.doc	Translation from Norwegian to German	Finished translating. Download result here.

Check out our other products:

LINGDYS er et program som gir skrivestøtte til dyslektikere og andre skrivesvake. Sliter du med å skrive, har du ofte behov for mer støtte enn en vanlig stavekontroll gir. Check it out here.

LINGRIGHT er utviklet for å være et hjelpemiddel for alle som må skrive engelsk på skolen, i jobben eller i privat sammenheng. Check it out here.

LINGSPEAK ARNE er en kunstig norsk stemme som LingIT har utviklet med støtte fra Microsoft Norge. Stemmen er basert på Telenor Talsmann. Check it out here.

Home Login Register About this site About LingIt Privacy policy

Copyright © LingIt AS 2009

APPENDIX C

Sprint 2 Appendix

Sequence diagram for translate

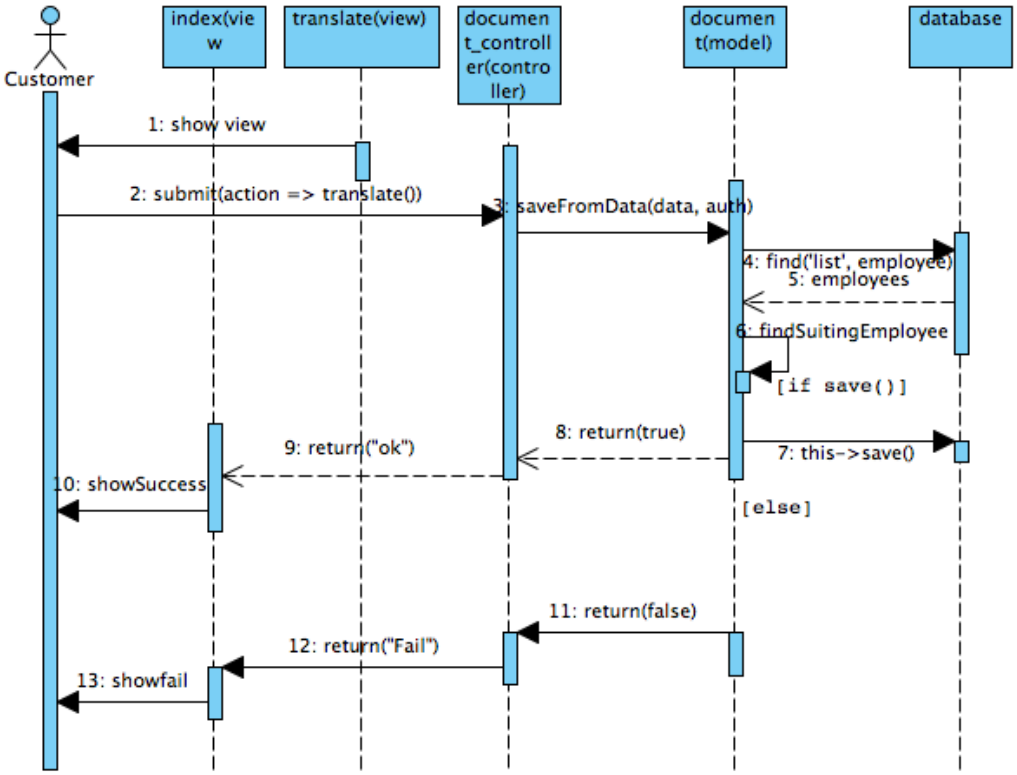


Figure C.1: Uploading a Document for Translating

APPENDIX D

Sprint 3 Appendix

Sequence diagram for translate

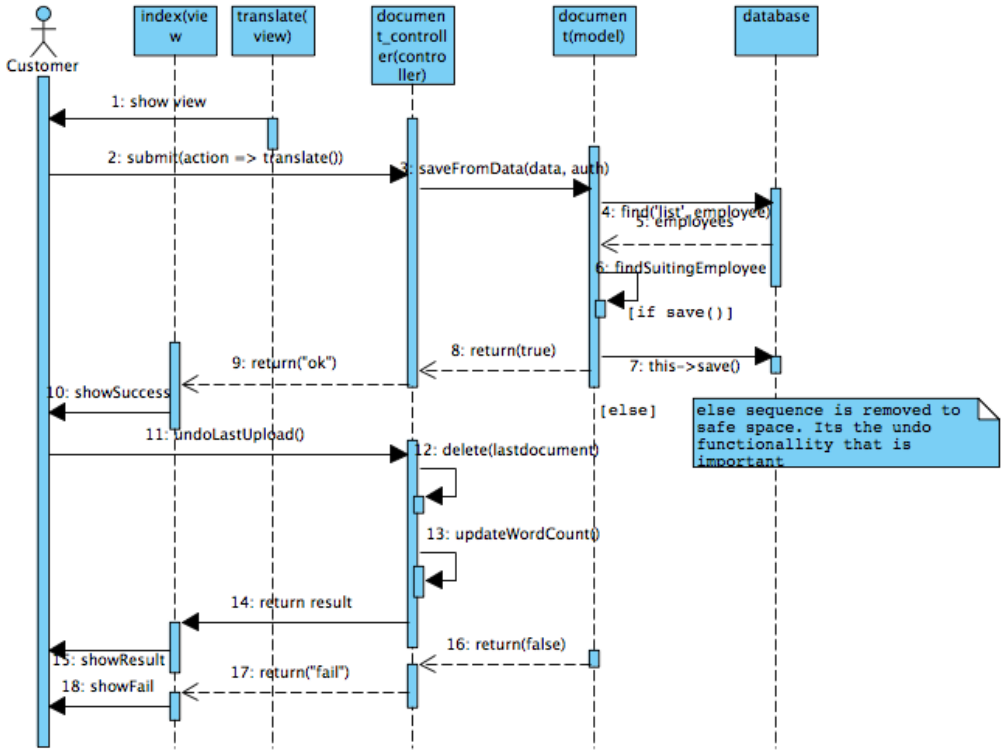


Figure D.1: Uploading a Document for Translating

APPENDIX E

Closure Appendix

E.1 Screen shots

This contains the screen shots of the finished design.

Admin page

This is the screen shots for the admin page.

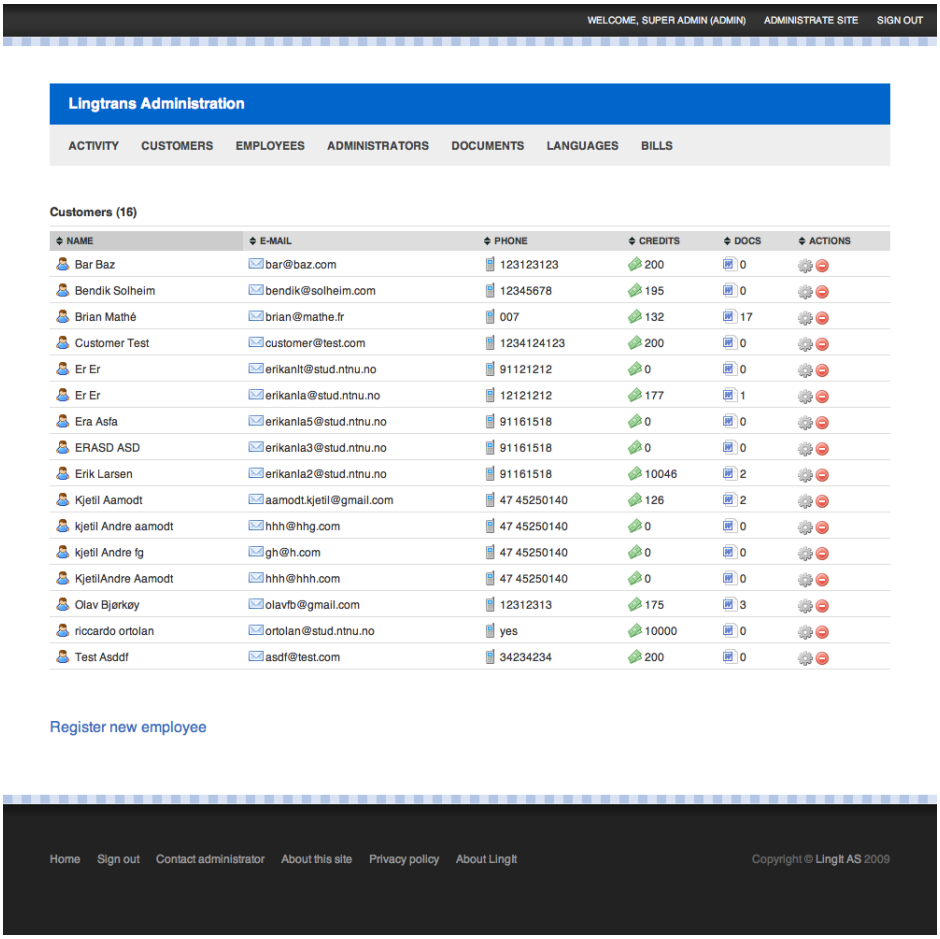


Figure E.1: Admin Page

E.1.1 Translate page

This is what the user see when he wants to upload a text for translation. The user can choose a file to upload or he can type the text in a text box. There are also some options for selecting translation language and response time for the translation.

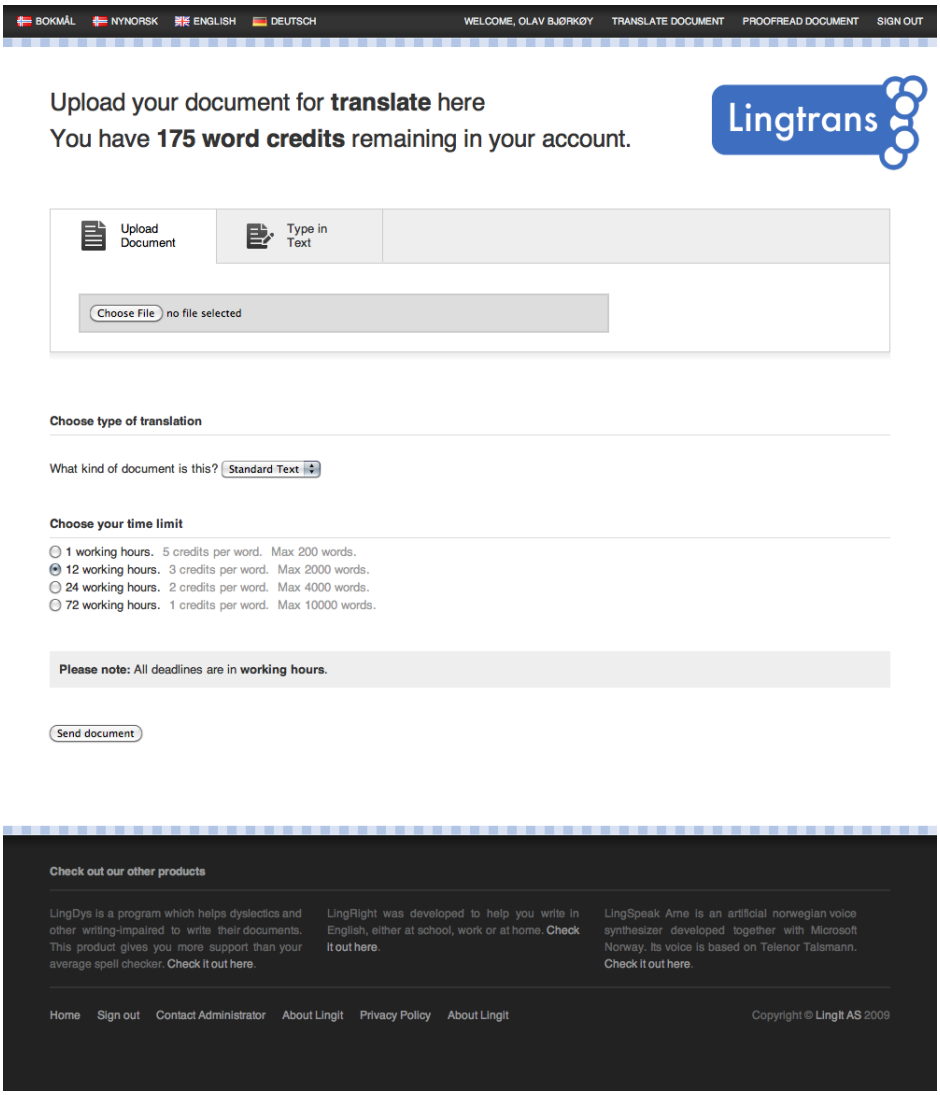

BOKMÅL
NYNORSK
ENGLISH
DEUTSCH
WELCOME, OLAV BJØRKØY
TRANSLATE DOCUMENT
PROOFREAD DOCUMENT
SIGN OUT
Upload your document for translate here
You have 175 word credits remaining in your account.
Lingtrans
Upload Document
Type in Text
Choose File
no file selected
Choose type of translation
What kind of document is this?
Standard Text
Choose your time limit
1 working hours. 5 credits per word. Max 200 words.
12 working hours. 3 credits per word. Max 2000 words.
24 working hours. 2 credits per word. Max 4000 words.
72 working hours. 1 credits per word. Max 10000 words.
Please note: All deadlines are in working hours.
Send document
Check out our other products
LingDys is a program which helps dyslectics and other writing-impaired to write their documents. This product gives you more support than your average spell checker. Check it out here.
LingRight was developed to help you write in English, either at school, work or at home. Check it out here.
LingSpeak Arne is an artificial norwegian voice synthesizer developed together with Microsoft Norway. Its voice is based on Telenor Talsmann. Check it out here.
Home
Sign out
Contact Administrator
About Lingit
Privacy Policy
About Lingit
Copyright © Lingit AS 2009

Figure E.2: Translate Page

E.1.2 Translator page

The translator will see this page when he logs in.

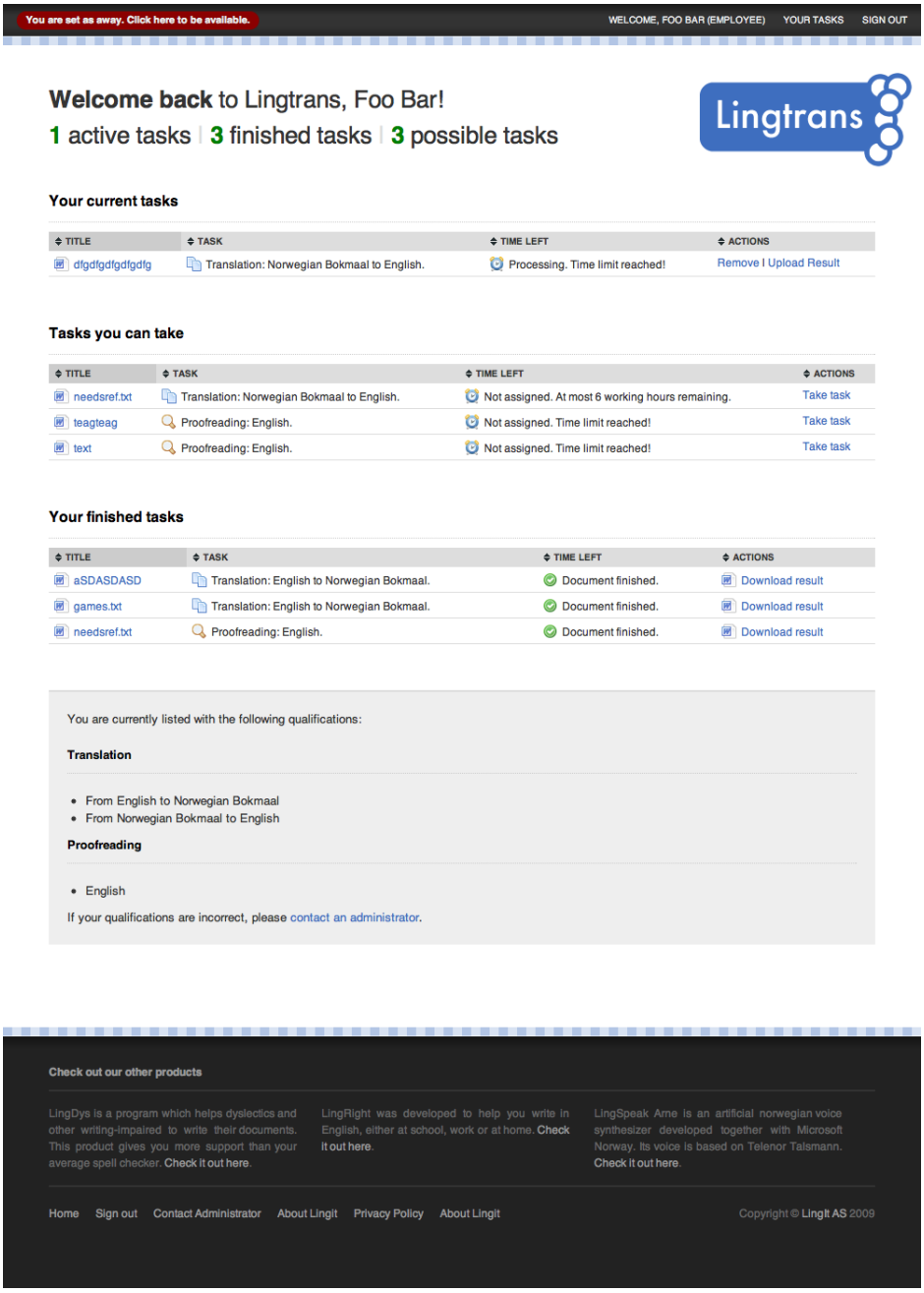


Figure E.3: Translator Page

E.1.3 User page

This is the page the user will see when he is logged in. He can choose between translate, proofreading and purchase of new words. The user will also see all previous documents.

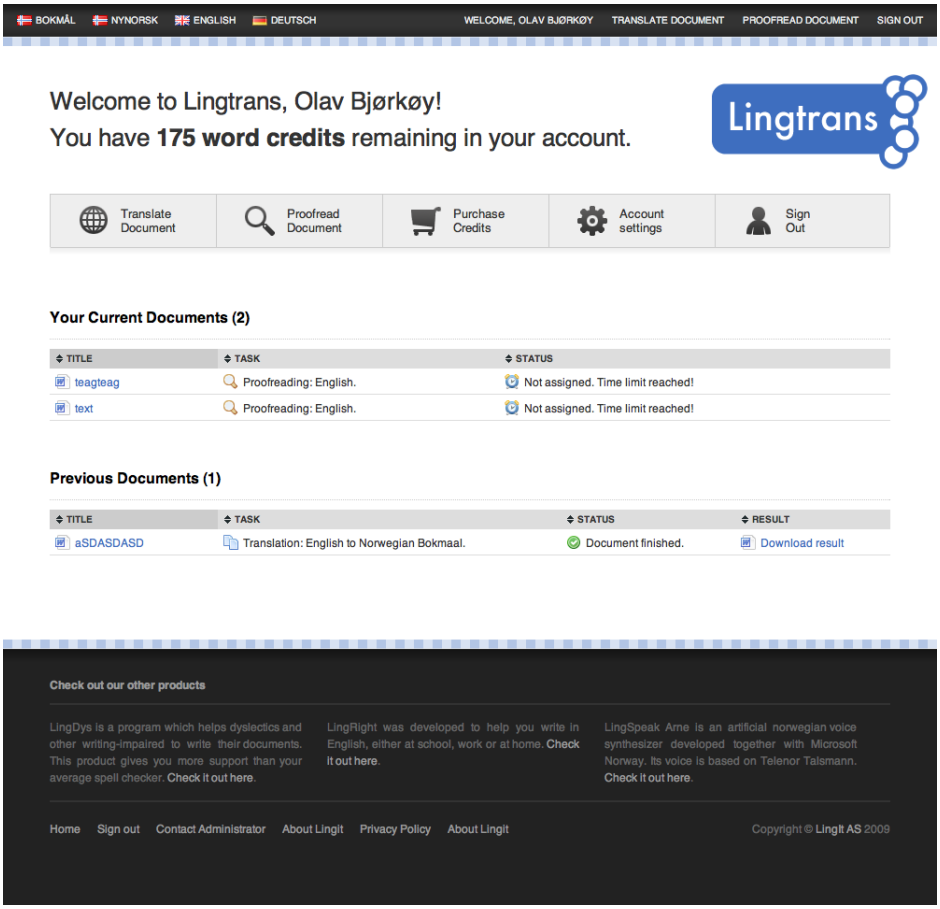


Figure E.4: Translator Page

APPENDIX F

Manuals

Here you will find the manuals to use the system. The section contains a manual for the customer, employee, administrator and deployment of the system. This manual will start from the next page.

Manuals and deployment

This document contains manuals for users of Lingtrans: Administrators, Employees and Customers. IT also contains a guide on how to set up the system on a server.

November 10, 2009

Contents

1 Deploying the System **1**
1.1 System Requirements 1
1.2 Required Server Tools 1
1.3 Setting up the database 2

2 Customer User Manual **3**
2.1 Web Application 3
2.2 MS Word add-in 3
2.2.1 Installing the Word add-in 3
2.2.2 Using the functions of the add-in 4

3 Employee User Manual **5**
3.1 Availability 5
3.2 Performing Work 5

4 Administrator User Manual **6**

CHAPTER 1

Deploying the System

This chapter explains how to set up Lingtrans on a server.

1.1 System Requirements

The system has been developed and tested on the following system:

- OS: Ubuntu Server Edition 9.04 (Jaunty)
- 256 MB RAM
- PHP 5.2.6-3ubuntu4
- MySQL Ver 14.12 distribution 5.0.75
- Apache 2.2

The system can be seen running at http://67.23.12.65:4466 (username: kpro, password: kpro123).

1.2 Required Server Tools

The system uses a few command line tools which must be installed on the server:

- sendmail (or equivalent) for php's mail() to work.
- poppler-utils to get the pdftotext utility.

The system does not currently perform virus checking on uploaded files. This could be done by using a system call to some tool, for instance `avgscan`. See the Document model file to find hooks for such a check.

1.3 Setting up the database

To set up the database, run the SQL setup code found in the file `sql.sql`. Please check the default rows added by this file before running it. Among other things, it contains the supported languages and three default user accounts, one for each type of user.

1.4 Customization

There is a few variables, like url and admin email that has to be set. These can be found at the top of the file `app/config/core.php`.

CHAPTER 2

Customer User Manual

There are two parts to the customer side of the system: The web application and the Microsoft Word addin.

2.1 Web Application

The first page the user sees is the index page, where the website and its features are presented. From here the user can either register as a new customer, or log in if this has been done before.

After registering, and before using the site, a new customer has to purchase a number of word credits to get access to all the site's features. This "welcome" screen where the user can download the MS Word addin and purchase word credits will be shown each time the user logs in until word credits are purchased.

When word credits have been purchased the user can use the web interface to upload documents for proofreading or translation. The available languages for each of these actions are dependant on the qualifications of the employees who are currently available. More on this in the next chapter.

2.2 MS Word add-in

2.2.1 Installing the Word add-in

The MS Word add-in can be downloaded as an archive file directly from the website main page. Its content then needs to be extracted into a folder. Then, just run the 'Setup' file and all the features will be installed next time Word is open.

2.2.2 Using the functions of the add-in

When installed, it should be added a ribbon named *Lingtrans* in the *Add-in* tab in MS Word.

You can start with checking the news of your account in pressing *Check news*. If there are some new documents available, a link will be displayed to the system homepage, where you will be able to download your finished document.

If you want to proofread a document, just push the *Proofread* button. Only the available languages will be displayed, corresponding to the employees currently available in our offices. If the language you wish is not displayed, you can try again later, waiting an employee to be available. Our policy is to have all the languages available at all times, so you shouldn't wait for too long.

The same logic is applied to the *Translation* functionnality: just push the button and you will be asked the preferences of your document. When selecting the language of your document, the list of languages to translate to will be updated at the same way than when proofreading, regarding what employees are available for these translations.

For the deadlines, you are warned the number of words that you can send within the deadline you select. You can compair it to the text you've selected to transfer to the service written at the top of the box.

2.2.3 Remarks for deployment

The add-in is guaranteed to be runable only on Office Word 2007 and has been tested only on Windows Vista, but should work properly on previous versions on Windows if run on Office 2007.

It has been developed with Visual studio 2008 as a VSTO add-in project. The target framework version of .NET is v3.5

There should be a re-check of the availability of languages when the user press OK on proofreading or translation (every time a document is sent to the API), in order to be sure that all languages are still available at that moment.

The username is not remembered from session to session. This can only be done in saving it in a seperate text file on shutdown of the add-in and read on startup. A beginning of function already creates a text document on add-in shutdown, in which the username is supposed to be saved. But due to lack of time, we couldn't understand why it didn't work.

CHAPTER 3

Employee User Manual

The employees of Lingtrans uses the website to take tasks and upload finished results.

Each employee is registered by an administrator, and their qualifications in different languages are also changed and updated by administrators. More on this in the next chapter.

3.1 Availability

An employee logs in from the same page as every other user. After logging in, the most important button is the one found in the top left hand corner of the page. This button toggles whether the employee is available for taking tasks or not. While the employee is "away", the system will not accept new uploads based on this employees qualifications, unless some other employee has the same skills.

For this reason, an employee should be set as available at all times when he or she can take tasks within a reasonable amount of time. The available flag is not reset when the employee logs out, so it's as important for the employee to set him or herself as away when no more tasks can be accepted.

3.2 Performing Work

When a new document is uploaded, the system will automatically assign it to the most appropriate available employee. If the system can't find a suitable employee, and if the current employee has the neccessary qualifications, the upload will show up under "available tasks" on the employee's home page.

After taking a task, it is shown under "your current tasks". Here, the employee can download the document, see the deadline, and the job that should be done. When the employee is finished, the finished result can be uploaded here as well. When the result is uploaded, the customer associated witht this document will be notified.

CHAPTER 4

Administrator User Manual

The admin can only be registered by other admins, and they log in through the same login page as everyone else.

When logged in, an admin is presented with a log of recent activity in the system. The rest of the admin pages are concerned with administering the different database tables - mostly deleting and editing entries.

The billing page is where all customer purchases are shown. These can eventually be marked as read, after the admin has sent the bill using Lingit's prefered method.

Admins are also responsible for setting the employee qualifications. This can be done by going to "employees" and editing one employee.

Everything that's not editable on the admin pages will have to be edited directly in the database.

For changing the localization strings of the customer pages, see the file `app/vendor/site_localization.php`.